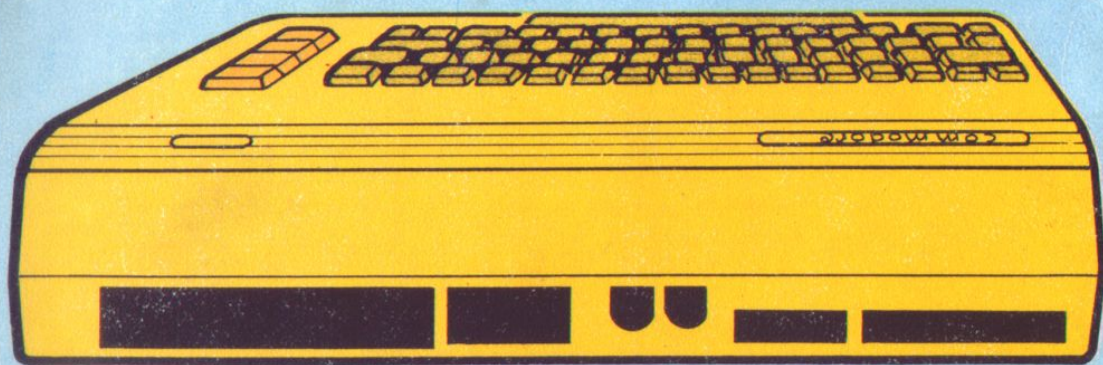
François Monteil

Come usare il tuo COMMODORE 64

Linguaggio macchina, ingressi/uscite e unità periferiche

Editrice Il Rostro

Come usare
il tuo
Commodore 64

2. Linguaggio macchina, ingressi/uscite e unità periferiche

Pubblicazioni di microinformatica della Editrice Il Rostro

H.J. Sacht - *I personal computer. Vademecum per chi vuole conoscerli, usarli, comprarli*

B.A. Artwick - *Interfacciamento dei microcomputer. Metodi e dispositivi*

F. Monteil - *Come usare il tuo Commodore 64*

F. e M. G. Monteil - *Come usare il tuo VIC 20*

J.Y. Astier - *Come usare il tuo Apple II*

Bernd Pol - *Come programmare in BASIC*

V. Cappelli - *Guida breve alla programmazione in BASIC*

R. Schomberg - *Il BASIC del tuo personal*

K.J. Schmidt e G. Renner - *Sistemi operativi per microcomputer: CP/M-CDOS-DOS*

I.R. Wilson e A.M. Addyman - *Introduzione al Pascal*

François Monteil

Come usare il tuo Commodore 64

2. Linguaggio macchina, ingressi/uscite e unità periferiche

Editrice Il Rostro

Indice

Prefazione

Questo libro si rivolge a tutti coloro che posseggono un Commodore-64, e costituisce il seguito di "Come usare il tuo Commodore 64" (1° volume).
Mentre il primo volume trattava soggetti quali il BASIC, i diversi modi di visualizzazione ed il sintetizzatore musicale, illustrandoli con numerosi esempi scritti in BASIC, il secondo volume si dedica in particolare al linguaggio macchina, alle funzioni di Input/Output e alle unità periferiche. Esso si indirizza quindi a quei lettori che, tramite la programmazione in BASIC, hanno potuto intravvedere le grandi possibilità offerte da questa macchina senza però poterle sfruttare completamente, a causa delle limitazioni di questo tipo di linguaggio.
Un primo capitolo descrive brevemente l'organizzazione materiale (hardware) del Commodore 64 ed in particolare la struttura della sua memoria.
Il secondo capitolo è interamente dedicato al linguaggio macchina. In esso imparerete come memorizzare delle routine in linguaggio macchina entro un programma BASIC e come salvarlo su cassetta o su disco. Inoltre una gran parte del capitolo è dedicata a descrivere le diverse routine, presenti nel sistema operativo, accessibili all'utilizzatore. Tali routine sono descritte singolarmente ed illustrate con numerosi esempi. Il tutto viene poi messo in pratica in un capitolo dedicato alla grafica in alta risoluzione ed agli SPRITE.
Imparerete come si possono visualizzare velocemente punti e curve sullo schermo, ed a programmare la forma ed il movimento degli SPRITE. Tutti gli argomenti descritti in questo capitolo sono illustrati con dei programmi originali, scritti in Assembler, che possono funzionare immediatamente sulla vostra macchina senza alcuna modifica. Un terzo capitolo affronta i problemi degli ingressi e delle uscite, con la descrizione del circuito di controllo speciale tipo 6526. Con il suo ausilio potrete realizzare facilmente dei sistemi di controllo automatico per la vostra casa, dei giochi luminosi, ecc. Inoltre viene descritto come realizzare un orologio in tempo reale assai preciso, che viene visualizzato in permanenza sullo schermo.
Un ultimo capitolo è dedicato alle periferiche ed ai programmi pronti disponibili per questo tipo di macchina.
Vogliamo comunque precisare che quest'opera non pretende di essere un corso sull'Assembler del 6510. Per conoscere maggiori dettagli su questo argomento, vi consigliamo la lettura di libri dedicati a questo microprocessore.

1
L'architettura del sistema

1.1 Generalità

Il Commodore 64 è quello che viene chiamato un "microcomputer" o "microelaboratore" individuale. Come tale, esso comprende tutti i componenti necessari per il suo funzionamento come sistema autonomo (a parte lo schermo TV, di cui abbiamo parlato nel 1° volume). Questo capitolo ha lo scopo di farvi comprendere il funzionamento generale della vostra macchina, anche senza entrare nei dettagli "elettronici" propriamente detti.

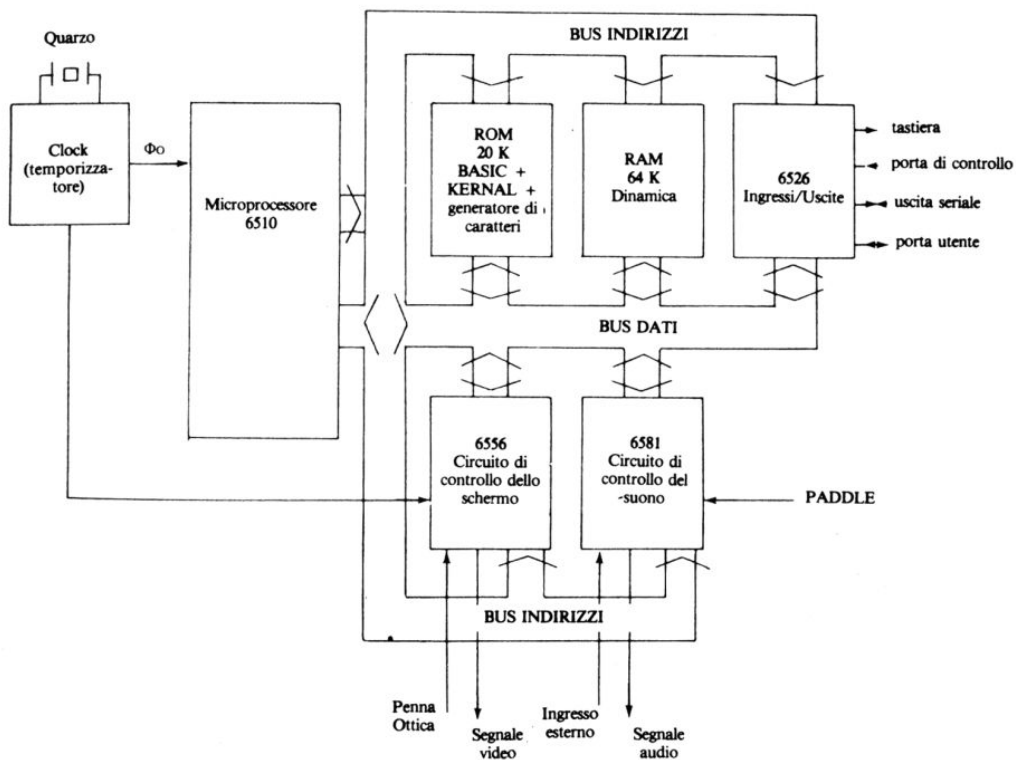
1.2 L'architettura del sistema

Il Commodore 64, come tutti i microcomputer, è costruito attorno ad uno di quegli "scarafaggi" a più zampe detti microprocessori. Nel nostro caso abbiamo a che fare con un 6510, derivato dal famoso 6502, che comprende una porta di INPUT/OUTPUT integrata.
Questo componente non ha ormai più bisogno di venire presentato, dato che su di esso si basano tutti gli attuali computer della gamma Commodore (PET, VIC 20, VBM, ...), nonché l'Apple II, l'Apple IIe, l'Apple III, l'Atom, l'ORIC I, e tanti altri...
Il 6510 è dotato esattamente del medesimo set di istruzioni del 6502. Come tutti i microprocessori, possiede un "bus" indirizzi, un "bus" dati ed un "bus" di controllo.
Questi bus permettono l'accesso alla memoria che contiene dei programmi (in BASIC o in linguaggio macchina). Questo può essere suddivisa in due sezioni: la memoria "morta" o accessibile solo in lettura (ROM = Read Only Memory); e la memoria "volatile" (RAM = Random Access Memory). La prima, come indica la denominazione inglese, può essere soltanto "letta", e contiene i programmi per la gestione del Commodore 64 (gestione della tastiera, dello schermo, degli INPUT/OUTPUT, ecc.), nonché l'Interprete BASIC ed il generatore di caratteri.

La seconda (memoria ad accesso casuale) contiene da un lato i programmi utente (in BASIC o in linguaggio macchina), ed inoltre la memoria per lo schermo che contiene i codici di tutti i caratteri visualizzabili.
Non intendiamo affrontare, come abbiamo già detto, l'illustrazione dei modi con cui il microprocessore "legge" e "scrive" nella memoria, né come viene effettuata la sincronizzazione ed il concatenamento di tutte le operazioni elementari necessarie per svolgere un compito relativamente complesso (come ad esempio il modo con cui viene visualizzato sullo schermo un dato carattere). Per soddisfare la vostra curiosità ed imparare qualcosa di più su tali argomenti vi suggeriamo di consultare delle opere specifiche (*). Qui ci accontenteremo di descrivere gli schemi di principio del sistema.

Oltre alla memoria dei due tipi che abbiamo citato, esistono dei componenti specializzati che al microprocessore appaiono come semplici locazioni di memoria, destinate a svolgere le operazioni di entrata (INPUT) e di uscita (OUTPUT): si tratta di circuiti tipo CIA (Complex Interface Adapter), nel nostro caso del modello 6526, di cui esistono due esemplari sul Commodore 64.
Torneremo su questo argomento più oltre nel libro, dedicandovi un intero capitolo. Questi circuiti integrati sono incaricati di gestire (tramite un ideoneo programma, s'intende) la tastiera, l'interfaccia seriale, la porta utente, e le porte di controllo destinate al collegamento dei PADDLE o dei JOYSTICK.
Oltre a questi, esiste un circuito specializzato, tipo 6566, che non è altro che il circuito di controllo dello schermo descritto nei dettagli nel 1° volume di quest'opera, ed il cui compito è di generare tutti i segnali necessari alla visualizzazione sullo schermo del monitor o del TV, in modo interamente (o quasi) "trasparente" per il microprocessore.
In modo alfanumerico ("testo") la memoria dello schermo contiene il codice del carattere che deve essere visualizzato, e che viene presentato all'ingresso di una memoria chiamata "generatore di caratteri". Questa memoria può essere di tipo RAM oppure ROM, come abbiamo visto nel 1° volume. Le informazioni contenute in questa memoria relative al carattere da visualizzare sono poi trasmesse al circuito di controllo dello schermo che si incarica di generare il segnale video composito a colori che ritroviamo infine sul connettore audio/video posto sul retro della macchina.

(*) Il lettore interessato potrà consultare l'elenco dei volumi di microinformatica della Editrice Il Rostro che appare all'inizio del libro.

Le informazioni relative al colore dei caratteri sono contenute in una memoria distinta, in forma di "parole" di 4 bit: tale memoria viene chiamata Memoria Colore.
Infine, esiste un circuito specializzato complesso, del tipo 6581, che costituisce il circuito di controllo dei suoni, anch'esso descritto nel 1° volume, con il compito di realizzare le funzioni sonore del Commodore 64.
Lo stesso circuito consente pure, grazie a due convertitori analogici/numerici (A/D) integrati, di leggere le posizioni assunte dai potenziometri contenuti nelle PADDLE collegate agli opportuni ingressi.
Lo schema di principio del Commodore 64 è il seguente:

Ed ora passeremo ad esaminare la "mappatura" del Commodore 64, vale a dire la collocazione nello spazio di memoria indirizzabile del 6510 di tutti i componenti di cui abbiamo parlato, ossia memoria, circuiti di I/O, circuiti di controllo dello schermo e del suono.

1.3 La mappa di memoria

Abbiamo detto che il microprocessore è dotato di un bus degli indirizzi. Questo è formato da 16 bit, come è comune per tutti i microprocessori ad 8 bit attualmente sul mercato. Esso può quindi indirizzare $2^{16} = 65536$ locazioni (celle) di memoria diverse, comprese fra gli indirizzi $0000 = 0 e $FFFF = 65535 (*).
In questo sottocapitolo esamineremo come questo spazio risulti suddiviso in funzione dei diversi componenti esistenti, e come possano variare i ruoli di talune di queste sezioni di memoria a seconda dei parametri programmabili dalla macchina.
Dato che il Commodore 64 dispone, come abbiamo visto, di 64 K byte di RAM, 20 K byte di ROM e di entrate/uscite (I/O) integrate, risulta evidente che le relative locazioni di memoria non possono venire tutte indirizzate contemporaneamente.
È a questo punto che interviene la porta di I/O integrata nel 6510. Grazie ai valori assunti da alcuni dei suoi bit, è possibile selezionare diverse combinazioni di ROM, RAM ed entrate/uscite, in funzione dell'impiego previsto. In particolare, la mappatura risulterà totalmente differente a seconda che si lavori, ad esempio:

— in BASIC (modo selezionato all'atto dell'accensione della macchina se non risulta collegata alcuna espansione)
— in regime CP/M (modo selezionato se sulla porta di espansione risulta collegata una cartuccia CP/M munita d'uno Z80).
— sotto il controllo di un programma applicativo in cartuccia ROM (gioco, programma per il trattamento di testi, ecc.).

La porta di entrata/uscita del 6510 è posta all'indirizzo $01 in pagina zero. I bit di tale porta destinati a controllare la configurazione dell'area di memoria devono venire configurati come uscita; il che si può fare ponendo ad 1 i bit corrispondenti al registro DDRA (Data Direction Register) situato all'indirizzo $00 in pagina zero.
Soltanto i tre bit più "bassi" (da 0 a 2) di questa porta vengono utilizzati per controllare la mappatura del Commodore 64. I bit da 3 a 5 sono destinati a comandare il funzionamento del lettore di cassette di nastro.

(*) N.d.T.: Ricordiamo che il simbolo $ premesso ad una coppia od un "quartetto" di cifre comprendenti anche i caratteri A-F indica che si tratta di un numero in notazione esadecimale (ad 1 o 2 byte).

Il ruolo di questi bit $0 \div 2$ è il seguente:

Bit 0: permette di selezionare la ROM o la RAM fra gli indirizzi $A000 e $BFFF. Normalmente questo bit è posto ad 1, ed allora seleziona la ROM Basic di 8 K.

Bit 1: permette di selezionare la ROM o la RAM fra gli indirizzi $E000 ed $FFFF. Normalmente posto ad 1, seleziona in tal caso la ROM Monitor (KERNAL) contenente il sistema operativo del Commodore 64.

Bit 2: permette di selezionare il generatore di caratteri in ROM oppure le entrate/uscite nell'area indirizzabile del 6510. L'abbiamo già incontrato nel capitolo relativo al circuito di controllo dello schermo del 1° volume: le entrate/uscite (CIA, controllore schermo e del suono) sono collocate nella medesima area di memoria del generatore di caratteri. Il bit 2 è normalmente posto ad 1, ed in tal caso seleziona le entrate/uscite. Il generatore di caratteri in ROM non è quindi più collocato nell'area indirizzabile del 6510, e soltanto il circuito di controllo dello schermo può accedervi per la visualizzazione. Quando tale bit invece è a 0, le entrate/uscite non sono più accessibili, ed il generatore di caratteri in ROM risulta nell'area indirizzabile del 6510. Si può allora ricopiare alcuni dei codici relativi ai caratteri ivi contenuti nella RAM, ossia programmare un proprio generatore di caratteri, secondo il metodo descritto nel 1° volume.

Un ultimo tipo di segnale (detto EXROM, ovvero "Estensione ROM") consente di selezionare la ROM o la RAM fra gli indirizzi $8000 e $9FFF. Questo segnale non corrisponde ad un bit della porta I/O del 6510, ma viene generato internamente alla macchina. Ed ora vi forniamo la "mappa della memoria" del Commodore 64 in modo di funzionamento standard, vale a dire in regime BASIC ma senza cartuccia di espansione collegata. È quella che si ottiene quando LORAM = HIRAM = CHAREN = EXROM = 1.

Indirizzo	Funzione
65535 = $FFFF	
	8K byte di ROM Monitor (KERNAL)
57334 = $E000	
	entrata/uscita utente n. 2 (disco)
57088 = $DF00	
	entrata/uscita utente n. 1 (CP/M)
56832 = $DE00	
	circuito CIA2 (bus seriale, porta utente)
56576 = $DD00	
	circuito CIA1 (tastiera)
56320 = $DC00	
	RAM colore
55296 = $D800	
	circuito di controllo per il suono
54272 = $D400	
	circuito di controllo dello schermo
53248 = $D000	
	4k byte di RAM (utile per memorizzare programmi in L.M.)
49152 = $C000	
	ROM BASIC 8K byte
40960 = $A000	
	RAM utente (memorizzazione programmi BASIC)
2048 = $0800	
	RAM schermo + puntatori degli SPRITE
1024 = $0400	
	zona di lavoro del sistema operativo e stack del 6510
0 = $0000	

Le altre configurazioni di memoria si possono facilmente dedurre da questa tramite quanto abbiamo detto in questo sottocapitolo.
Ora che conoscete un po' l'architettura del vostro microelaboratore, possiamo passare ad un soggetto molto interessante: l'accesso al linguaggio macchina, che formerà l'oggetto del capitolo che segue.

2
L'accesso al linguaggio macchina

2.1 Introduzione

Sinora abbiamo parlato esclusivamente di programmazione in BASIC sul Commodore 64 (volume 1°). Questo linguaggio, assai semplice e facilmente assimilabile, risulterà senz'altro sufficiente per permettervi di scrivere la maggior parte dei vostri programmi — siano questi di tipo scientifico, grafico o sonoro. Certo che se pensate di scrivere dei programmi di giochi di movimento veloci, come quelli che sono disponibili come cartucce di espansione Commodore, vi accorgerete ben presto delle forti limitazioni di questo tipo di linguaggio in termini di velocità. In pratica, l'interprete BASIC, quando incontra una linea di istruzioni del programma, la «traduce» in istruzioni «macchina», che vengono poi eseguite dal microprocessore. Così, ad esempio, le varie istruzioni comprese entro un ciclo FOR... NEXT verranno interpretate di nuovo tante volte quante vengono eseguite, e ciò comporta un discreto rallentamento.
Inoltre, nel corso dell'esecuzione di un programma l'interprete BASIC crea diverse zone entro cui memorizza le variabili semplici, le variabili multiple e le stringhe di caratteri. Ogni volta che incontra uno di questi elementi, esso deve andare alla ricerca del relativo valore scorrendo tali zone, oppure deve aggiungere il nuovo elemento nella zona relativa. Perciò, dato che la zona di memorizzazione delle variabili semplici è posta in memoria prima di quella delle variabili multiple, l'interprete deve provvedere a spostare interamente quest'ultima ogni volta che incontra una nuova variabile semplice: con altra perdita considerevole di tempo se il numero delle variabili multiple è relativamente elevato.
Se l'unico problema che vi pone il BASIC è la velocità di esecuzione, potete raggiungere dei risultati applicando le regole seguenti:

— utilizzate numeri di linea piccoli, in modo da risparmiare in area di memoria occupata

— ponete più istruzioni su di una stessa linea (istruzioni multiple)
— sopprimete le linee con i commenti (REM)
— eliminate gli spazi
— diminuite il numero di variabili multiple e non eccedete nei valori delle relative dimensioni
— non accrescete eccessivamente il numero delle variabili
— utilizzate al massimo le variabili intere; ecc.

Non pretendiamo che questa lista di suggerimenti sia completa: esistono molti altri "trucchi" per abbreviare il tempo di esecuzione di un programma. Se quanto sopra non basta, o se il vostro problema è un altro, dovrete rassegnarvi a penetrare nell'"Assembler del 6502".
La cosa sarà necessaria se desiderate programmare direttamente le vostre entrate/uscite tramite il circuito CIA tipo 6526, od anche, semplicemente, quando avrete bisogno di una funzione che non è prevista dal normale interprete BASIC.
Potrete allora rendervi rapidamente conto dei vantaggi della programmazione in Assembler, fra i quali:

— forte guadagno di velocità
— forte risparmio nell'occupazione di memoria rispetto ad un programma BASIC equivalente (se esiste).

Prima di progredire nella lettura di questo capitolo, occorre che impariate a conoscere le istruzioni del 6510, che sono poi identiche a quelle del 6502.
Dato che quest'opera è dedicata al Commodore 64, e non al 6510, ci accontenteremo di fornirvi in Appendice una tabella del gruppo di istruzioni di questo microprocessore.

2.2 Accesso a partire dal BASIC

Nel capitolo del 1° volume dedicato al BASIC abbiamo intenzionalmente lasciato un po' da parte le istruzioni SYS ed USR. Sono quelle che permettono di richiamare dei sottoprogrammi scritti in linguaggio macchina (L.M.) a partire da un programma scritto in BASIC.
Vi renderete presto conto che esse costituiscono il mezzo generale e più comodo per accedere al linguaggio macchina sul Commodore 64. Esso permette di approfittare dei vantaggi che abbiamo già citato nel paragrafo precedente mantenendo la comodità d'impiego propria del BASIC.

In effetti, risulta comodo poter disporre di istruzioni di INPUT/OUTPUT "belle e pronte" quali INPUT, PRINT, GET, ..., che risultano altrimenti alquanto scomode da programmare in L.M.

2.2.1 L'istruzione SYS

La sintassi di questa istruzione è la seguente:

```
SYS (indirizzo)
```

dove "indirizzo" è un valore decimale compreso naturalmente fra 0 e 65535, ossia fra i valori indirizzabili del 6510.

Esempio:

```
10 SYS(5000)
```

Quando l'interprete BASIC incontra un'istruzione SYS, esso fa saltare il 6510 ad un sottoprogramma il cui indirizzo iniziale è quello compreso fra le parentesi (nel nostro caso: 5000), che viene poi eseguito partendo da tale indirizzo.
Quando il 6510 incontra l'istruzione di rientro dal sottoprogramma RTS (op-code $60 = 96), si ha il ritorno automatico al BASIC, e l'esecuzione del resto del programma BASIC stesso.
Osserviamo che alla fine della routine chiamata da SYS è assolutamente necessaria la presenza dell'istruzione RTS. In caso contrario, il vostro Commodore 64 rischia seriamente di bloccarsi ("crash").
Questa istruzione SYS costituisce quindi un comodo mezzo per accedere al linguaggio macchina a partire dal BASIC.
Essa però non permette di passare direttamente dei parametri al programma in L.M.; un tale passaggio si dovrà realizzare se necessario preventivamente tramite delle istruzioni POKE nelle locazioni opportune.

Esempio di impiego: Se battete sulla tastiera SYS (64738) e RETURN, ottenete la reinizializzazione completa ("reset") del vostro Commodore 64, con la comparsa sullo schermo del ben noto messaggio

```
    **** COMMODORE 64 BASIC V2 ****
64 K RAM SYSTEM 38911 BASIC BYTES FREE
READY
```

Se, prima di eseguire un simile comando diretto, avevate un programma BASIC in memoria, ebbene, l'avrete perso del tutto.
Facciamo ora l'ipotesi che si desideri ottenere un breve ritardo mediante una routine in linguaggio macchina.
Ecco la routine:

```
C000 > LDX#FF      ; inizializza il contatore dei cicli
C002 > DEX         ; è trascorso il tempo richiesto?
C003 > BNE FD      ; no, ritorna da capo
C005 > RTS         ; si, ritorna al programma principale
```

Per attivarlo dal BASIC si scriverà:

```
10 REM GENERAZIONE DI UN RITARDO
20 SYS(49152)
30 END
```

Nelle pagine che seguono vedremo come si fa per introdurre questo programma in L.M. nella memoria del computer.

2.2.2 L'istruzione USR

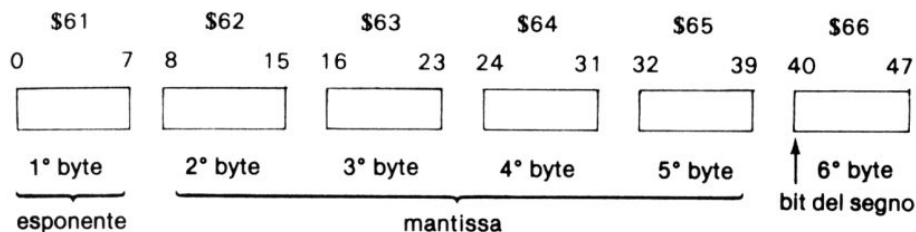
L'istruzione USR si distingue da quella SYS nel senso che essa permette di passare un parametro fra il programma BASIC ed il programma in L.M. La sintassi di questa istruzione è:

```
USR(X)
```

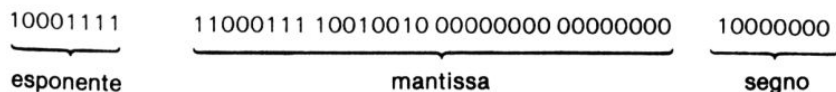
dove X è il parametro da passare al programma in L.M.
Quest'ultimo dovrà, come per il caso di quello richiamato con SYS, terminare con un'istruzione RTS.
Come potete constatare, l'istruzione USR non specifica in alcun modo l'indirizzo d'inizio del programma in L.M. Questo indirizzo infatti dovrà essere stato preventivamente memorizzato agli indirizzi 785 = \$0311 e 786 = \$0312, che conterranno rispettivamente nell'ordine il byte "basso" e quello "alto" dell'indirizzo di inizio della routine.
Il parametro passato viene collocato in un accumulatore posto agli indirizzi da 97 = \$61 e 102 = \$66 in pagina zero. Tale parametro viene memorizzato in formato di numero reale in virgola mobile, in maniera un po' diversa però da quella che già conosciamo.

Vediamo meglio come avviene questa codifica:

— l'esponente viene memorizzato all'indirizzo 97 = \$61, secondo lo stesso metodo esposto nel capitolo relativo al BASIC del 1° volume di quest'opera (notazione con spostamento).
— la mantissa viene memorizzata in 4 byte (da 98 = \$62 a 101 = \$65). Qui non compare il segno della mantissa, ed il bit più basso del 1° byte (indirizzo 98) rappresenta il bit che sino allora era sottinteso, che fornisce una mantissa in ogni caso compresa fra 0.5 ed 1.
— il segno della mantissa è invece dato dal bit più "alto" del byte di indirizzo 102 = \$66.

Si ha dunque la seguente rappresentazione:

$$(\text{Numero}) = (a_8 2^{-1} + a_9 2^{-2} + a_{10} 2^{-3} \ldots + a_{38} 2^{-31} + a_{39} 2^{-32})\ 2^{(\text{esponente} - 128)}$$

Così, il numero —255.45 E2 risulta codificato in questo modo:

10001111	11000111 10010010 00000000 00000000	10000000
esponente	mantissa	segno

Quando in un programma BASIC si scrive B = USR(A), il valore di A viene trasferito nell'accumulatore di cui abbiamo appena parlato; poi il programma in L.M. esegue il trattamento desiderato su questo valore A.
Poi, quando il 6510 trova l'istruzione RTS, si ha il ritorno al programma BASIC e la variabile B avrà come valore il risutlato delle elaborazioni effettuate dal L.M. sul parametro A.

Vediamo meglio con qualche esempio:
Impostate il seguente programma:

```
 5 POKE 4400,96
10 POKE 785,48
20 POKE 786,17
30 INPUT A:N = USR(A) : PRINT N
```

e date il RUN.
Qualsiasi sia il valore introdotto poi dalla tastiera, il risultato visualizzato sullo schermo sarà eguale a questo. La cosa si spiega semplicemente così: l'istruzione USR chiama la routine in L.M. che principia all'indirizzo 48 + 256×17=4400. Questa routine in realtà è costituita unicamente dall'istruzione RTS (op-code 96). A non viene quindi modificato.
Ed ora provate ad impostare le linee seguenti:

```
5 POKE 4400,198
6 POKE 4401,97
7 POKE 4402,96
```

che corrispondono alla routine in L.M. seguente:

```
1130 > DEC61      ; A = A/2
1132 > RTS
```

L'istruzione DEC 61 decrementa di 1 l'esponente, e la cosa equivale a dividere il numero per 2. Se fate marciare il programma completo (linee 5, 6, 7, 10, 20, 30) il risultato visualizzato sarà il valore impostato dalla tastiera diviso per 2. Così, se impostate 22 vedrete comparire 11.
Abbiamo così visto come si può modificare l'esponente del valore A. Ovviamente, avremmo potuto modificare in modo simile la mantissa, oppure il segno od anche tutte e tre questi elementi assieme.
In pratica, per la codifica relativamente complessa dei numeri in virgola mobile, l'impiego dell'istruzione USR sarà riservata a casi particolari (calcolo scientifico, ad es., se la velocità è un fattore critico e se si richiede un'elaborazione particolare).
Inoltre, come visto l'istruzione USR non consente che il passaggio di un solo parametro, e la cosa può in certi casi non bastare.
Sarà allora sempre possibile utilizzare l'istruzione SYS accompagnata, per il passaggio di uno o più parametri, dalle opportune istruzioni POKE precedenti. Occorrerà tuttavia accertarsi sempre che i dati "pas-

sati" al programma in L.M. non vengano memorizzati nella zona riservata al BASIC od al Monitor.

2.3 I mezzi per la programmazione in linguaggio macchina

Avere l'intenzione di programmare in linguaggio macchina è una cosa; riuscire a realizzarlo in modo facile ed efficiente è un'altra cosa. In questa sezione del capitolo cercheremo di tentare un'illustrazione dei concetti presentando i vari mezzi disponibili al programmatore per scrivere delle routine in L.M.

1.a soluzione

Consiste nell'introdurre direttamente in memoria i vari codici operativi (op-code), gli operandi ed i dati tramite delle istruzioni POKE. L'unico problema sta nel fatto che l'istruzione POKE non accetta che valori in notazione decimale, mentre le istruzioni macchina del 6510 sono generalmente sotto forma esadecimale o come mnemonici.
Prima di operare nel modo indicato occorre quindi effettuare la conversione da esadecimale a decimale, il che alla lunga può risultare ben fastidioso. Un simile metodo non sarà quindi utilizzabile che nel caso in cui ricorriate raramente alle routine in linguaggio macchina, e per di più quelle che scrivete siano in genere corte.

2.a soluzione

Si può adottare quando si dispone di un programma che permette di caricare direttamente in memoria, di rileggere e di modificare dei byte, direttamente in esadecimale. Il programma potrà essere associato ad un "assemblatore" linea-linea (il cui compito è di tradurre opportunamente i codici mnemonici delle istruzioni del 6510), di un "disassemblatore" e di un "debugger" che permetta di verificare passo a passo il programma in L.M. Tutte queste caratteristiche ed altre ancora sono state da noi incluse in un programma, messo appositamente a punto per il Commodore 64, che viene messo in commercio, sotto forma di cassetta, in questa stessa collana, col titolo "Monitor in linguaggio macchina per il Commodore 64" (*)

(*) N.d.T. Ovviamente gli Autori si riferiscono al mercato francese: ma non vi è dubbio che programmi simili siano già o stiano per essere disponibili anche in Italia.

Nel seguito di questo libro, tutti i listati in linguaggio macchina saranno forniti usando la medesima forma di presentazione utilizzata (sullo schermo) da questo programma, in modo da non disorientare il lettore meno esperto.
Osserviamo che questo programma potrà costituire un eccellente ausilio per tutti coloro che vogliono sviluppare dei sottoprogrammi di dimensioni ragionevoli (qualche centinaio di byte al massimo). Per coloro che intendono scrivere dei programmi "monstre" di parecchi Kbyte si impone la soluzione n. 3.

3.a soluzione

Consiste nell'avere a disposizione un vero e proprio "Editor-Assembler". Per essere impiegabile in pratica un simile programma dovrà essere su disco, richiedendo quindi l'impiego di un lettore di dischi, come il VIC 1541. Detto questo, va precisato che esso consente solo l'introduzione e l'assemblaggio del programma, per cui per la sua messa a punto necessita dell'utilizzo di un monitor-"debugger".

2.4 Memorizzazione dei programmi in linguaggio macchina

In questa sezione illustreremo come si può memorizzare una routine in linguaggio macchina, e come si può inoltre salvarlo su cassetta o su dischetto.

2.4.1 Collocazione in memoria

Una routine in L.M. dovrà in ogni caso venire memorizzata in una zona in cui non possa subire interferenze da parte del sistema operativo del Commodore 64 o da un eventuale programma BASIC in corso di esecuzione.
Pertanto, dovranno senz'altro escludersi gli indirizzi da 0 a 2048, anche se talune di queste locazioni sono disponibili, come vedremo più avanti. Infatti, questa area di memoria contiene la pagina zero (indirizzi da 0 a 255) utilizzata dal sistema operativo; lo "stack" del 6510 (indirizzi da 256 a 511); un'altra zona dove lavora il Monitor (indirizzi da 512 a 1023); e la memoria dello schermo (indirizzi da 1024 a 2047). I programmi BASIC sono poi memorizzati a partire dall'indirizzo 2048.

Inoltre, le variabili semplici e multiple sono memorizzate immediatamente dopo il programma BASIC, mentre le stringhe di caratteri sono memorizzate a partire dall'indirizzo 40959, procedendo verso il basso della memoria (indirizzi inferiori). Se si desidera memorizzare i programmi in Assembler nella zona di lavoro dell'interprete BASIC, si può ricorrere a soluzioni diverse che ora esamineremo.

1. Memorizzazione nella parte alta della RAM BASIC

Si tratta di memorizzare il o i programmi in L.M. nella zona inizialmente riservata alla memorizzazione delle stringhe di caratteri. Niente di più semplice: nel 1° volume abbiamo descritto le funzioni dei puntatori alla fine della memoria ed alla zona destinata a ricevere le stringhe.
Se spostiamo i valori di questi due puntatori verso il basso, l'inteprete BASIC assumerà che lo spazio di memoria disponibile si sia ridotto, ma per il resto seguiterà a funzionare normalmente. Il puntatore alla fine della memoria si trova agli indirizzi 55 = $37 e 56 = $38, mentre il puntatore all'inizio della zona di memoria riservata alle stringhe si trova agli indirizzi 51 = $33 e 52 = $34. Se il nuovo valore da assegnare ad uno di questi due puntatori è "INDIRIZZO" (il valore di INDIRIZZO dovrà naturalmente essere inferiore a 40959), basterà, per proteggere la zona compresa fra INDIRIZZO e la locazione 40959, impostare sulla tastiera le linee seguenti:

```
POKE 51, INDIRIZZO —256 * INT (INDIRIZZO/256)
POKE 52, INT (INDIRIZZO/256)
POKE 55, INDIRIZZO —256 * INT (INDIRIZZO/256)
POKE 56, INT (INDIRIZZO/256)
```

Esempio: per la memorizzazione di una routine lunga 200 byte fra gli indirizzi 40760 e 40959 basterà impostare:

```
POKE 51, 56
POKE 52, 159
POKE 55, 56
POKE 56, 159
```

2. Memorizzazione nell'area BASIC della RAM

In questo caso, la routine in L.M. viene caricata immediatamente di se-

guito al programma BASIC residente in memoria. La procedura da seguire in tal caso è questa: prima di tutto, introdurre come al solito in memoria il vostro programma BASIC dalla tastiera; una volta che esso è stato messo a punto, impostare sulla tastiera:

```
PRINT PEEK(45) + 256*PEEK(46)
```

dove 45 = $20 e 46 = $2E sono gli indirizzi del puntatore all'inizio della zona di memoria riservata alle variabili. Con il comando diretto citato ricaverete quindi il valore dell'indirizzo iniziale di tale area.
Per proteggere una parte della RAM basta dunque spostare verso l'alto il numero di byte necessari.
Proseguendo l'esempio precedente:

l'INDIRIZZO è dunque il valore di PEEK(45) + 256*PEEK(46)

Ed ora si imposterà sulla tastiera:

```
POKE 45, INDIRIZZO + 200 — 256*INT ((INDIRIZZO + 200)/256)
POKE 46, INT((INDIRIZZO + 200)/256)
```

Se questo metodo presenta il vantaggio di generare un programma BASIC ed uno in L.M. in un blocco unico (la parte di memoria compresa fra il puntatore all'inizio della RAM ed il puntatore all'inizio dell'area delle variabili), ha però l'inconveniente che non permette più di modificare agevolmente il programma BASIC.
Infatti, l'aggiunta anche di una singola linea significa la distruzione parziale o totale della routine in L.M.
C'è sempre una soluzione anche per questi casi, che consiste nello spostamento del programma in Assembler prima di ogni modifica al programma BASIC, però ciò può comportare il rischio che il primo non funzioni più correttamente se in esso sono presenti dei salti "assoluti" (istruzioni JMP) a punti posti al suo interno. Nella maggior parte dei casi, perciò, lo spostamento della routine in L.M. richiederà un suo riassemblaggio preventivo. (La cosa può però essere evitata quando nel programma siano previsti solo dei salti "relativi" o dei salti verso indirizzi esterni).
Ovviamente, con lo stesso metodo opportunamente adattato è possibile pure memorizzare una routine in L.M. in una zona qualsiasi della RAM BASIC, ma il tutto a vostro rischio e pericolo. La cosa sarà possibile più facilmente se il programma BASIC è molto corto e non si rischia di allocare variabili semplici o multiple o valori dei tasti nella zona

in cui è impiantata la routine in L.M.
Solitamente, se la vostra routine "marcia" da sola, vale a dire non è richiamata da un programma BASIC, non ci sono più problemi, e potrete memorizzarla dove più vi piaccia nella RAM.

3. Una zona di memoria privilegiata

Nella sua configurazione base, il Commodore 64 (al momento dell'accensione della macchina) presenta la caratteristica di disporre di un'area di memoria RAM non accessibile all'interprete BASIC, come abbiamo potuto vedere esaminando la mappa della memoria.
Questa area di memoria è collocata fra gli indirizzi $C000 = 49152 e $CFFF = 53247. Sarà dunque possibile memorizzare in questa area della routine in L.M. senza alcun rischio che venga danneggiata, alla sola condizione che non superi le dimensioni accennate, ossia non sia più lunga di 4 Kbyte.

2.4.2 Salvataggio delle routine in linguaggio macchina

Ora che sapete dove potete memorizzare questo tipo di programmi, rimane il problema di come conservarli su cassetta o su disco. Infatti, vi dovreste essere accorti che il BASIC del Commodore 64 non dispone di istruzioni del tipo BLOAD o BSAVE che permettono di caricare o di salvare dei file "binari" (B), come i programmi in L.M. Le normali istruzioni LOAD e SAVE operano solo su dei programmi BASIC. Nei paragrafi seguenti vedremo alcuni metodi disponibili per il salvataggio delle vostre routine in Assembler.

1. Dissimulazione entro dei DATA

Questo è probabilmente il più semplice metodo a disposizione per salvare e ricaricare delle routine in L.M. richiamate da un programma principale in BASIC. In cambio, risulta probabilmente il più complicato dal punto di vista della scrittura della relativa routine, perché la stessa deve essere espressa in decimale (vedi par. 3).
Il metodo consiste nel raggruppare in una o più istruzioni DATA i codici operativi (op-code), gli operandi ed i dati che costituiscono la routine in L.M. Quando il programma BASIC viene lanciato da un RUN, que-

sti vari byte vengono caricati nella memoria mediante istruzioni POKE. Successivamente, la routine memorizzata a partire da un certo indirizzo può venire richiamata con un'istruzione SYS oppure USR.
Esempio: Riprendiamo la routine che permette di generare un breve ritardo:
sarà sufficiente scrivere il seguente programma BASIC:

```
10 DATA 162, 255, 202, 208, 253, 96
20 FOR I = 49152 TO 49157
30 READ A
40 POKE I,A:NEXT I
50 SYS (49152)
```

in cui

```
162 = $A2      (istruzione LDX)
255 = $FF
202 = $CA      (istruzione DEX)
208 = $D0      (istruzione BNE)
253 = $FD
 96 = $60      (istruzione RTS)
```

Le linee da 10 a 40 permettono di caricare in memoria (nella regione non accessibile all'interprete BASIC) la routine in linguaggio macchina. La linea 50 serve a lanciare questa routine, e quindi a generare il ritardo corrispondente.
In pratica, questo metodo si presta bene solo per routine molto corte (al massimo qualche dozzina di istruzioni).

2. Salvataggio assieme al programma BASIC

Quando si esegue un SAVE, il contenuto della zona di memoria posta fra gli indirizzi indicati dai puntatori all'inizio della RAM disponibile (puntatore all'inizio dell'area BASIC) ed all'inizio della zona di memoria delle variabili viene trasferito sul nastro della cassetta o sul disco. I vari byte che sono presenti in questa zona di memoria vengono inviati serialmente all'uscita dell'interfaccia per le cassette, od alla porta seriale. Il sistema operativo del Commodore 64 non fa differenze sul contenuto di questa memoria (sia programma BASIC, od in L.M., ecc...), considerandolo semplicemente come una successione di byte di 8 bit. Se pertanto si memorizza una routine in L.M. fra gli indirizzi IND1 e

IND2, e si provvede a spostare i due puntatori citati in modo da inquadrare questa area di memoria, è possibile salvare la citata routine mediante un SAVE, e ricaricarla poi con un LOAD. Osserviamo che questo modo di procedere funziona su tutte le estensioni di RAM accessibili al BASIC.

Esempio: Si abbia una routine in L.M. posta fra gli indirizzi

IND1 = $8120 = 33056   e   IND2 = $81E7 = 33255

Il puntatore all'inizio del BASIC è posto agli indirizzi 43 = $2B e 44 = $2C; si imposterà allora sulla tastiera

```
POKE 43, IND1—2—256*INT((IND1—2)/256)
POKE 44, INT((IND1—2)/256)
```

Analogamente, il puntatore all'inizio dell'area delle variabili è situato agli indirizzi 45 = $2D e 46 = $2E; perciò si batteranno le linee

```
POKE 45, IND2+1—256*INT((IND2+1)/256)
POKE 46, INT((IND2+1)/256)
```

I numeri 2 ed 1 derivano dal fatto che il puntatore all'inizio del BASIC è fissato 2 byte avanti all'inizio del programma in L.M., e il puntatore all'inizio dell'area delle variabili immediatamente dopo di esso.
Per salvare questa routine basterà impostare sulla tastiera

```
SAVE "PROGRAMMA",1,1
```

per il salvataggio su cassetta (l'1 in questo comando permetterà di ricaricare la routine al medesimo indirizzo dal quale fu prelevata all'atto del salvataggio, senza la qual cosa ci sarebbe il rischio che non funzioni più); e

```
SAVE "PROGRAMMA",8,1
```

per il salvataggio su disco. Per ricaricare la routine in memoria si farà

LOAD "PROGRAMMA",1,1 per il caricamento da cassetta, o
LOAD "PROGRAMMA",8,1 per il caricamento da disco

Dopo il SAVE si potrà naturalmente applicare anche il comando VERIFY come nel caso di un salvataggio di programma BASIC: si scriverà allora

VERIFY "PROGRAMMA",1 oppure
VERIFY "PROGRAMMA",8

Questo metodo può essere impiegato per salvare una routine in L.M. da sola, oppure assieme ad un programma BASIC, dopo l'opportuno spostamento dei puntatori.

3. Utilizzo dei sottoprogrammi del sistema

Questo metodo ricorre a delle routine contenute entro la ROM del Monitor (KERNAL). Esso consente di salvare o di caricare programmi su cassetta oppure su disco. L'accesso a queste routine, contrariamente a quanto si è visto finora, si può fare solo per il tramite del linguaggio macchina.
Queste routine verranno descritte un po' più oltre in questo capitolo, e perciò vi rimandiamo sin d'ora al paragrafo intitolato "I sottoprogrammi del sistema".

2.5 Qualche indirizzo utile

Questa sezione descrive il ruolo e la funzione svolti da alcune locazioni di memoria impiegate dal sistema operativo del Commodore 64.
Alcune di queste le abbiamo già incontrate (come i puntatori già citati), altre risulteranno nuove.

1. $0000 (=0)
   Registro DDR (Data Direction Register) della porta I/O integrata nel 6510.
2. $0001 (=1)
   Porta di entrata/uscita (I/O) del 6510.
3. $0002 (=2)
   Non utilizzata dal sistema operativo. Essa è quindi disponibile, ad esempio per il passaggio di un parametro ad un programma in L.M.
4. $002B-$002C (=43—44)
   Puntatore all'inizio del BASIC.

5. $002D—$002E (=45—46)
   Puntatore all'inizio della zona di memoria delle variabili.
6. $002F—$0030 (=47—48)
   Puntatore all'inizio della zona di memoria delle variabili multiple.
7. $0031—$0032 (=49—50)
   Puntatore alla fine della zona di memoria delle variabili multiple.
8. $0033—$0034 (=51—52)
   Puntatore della zona di memoria delle stringhe di caratteri.
9. $0037—$0038 (=55—56)
   Puntatore alla fine della RAM disponibile.
10. $0039—$003A (=57—58)
    Numero della linea di istruzioni BASIC corrente.
11. $003F—$0040 (=63—64)
    Puntatore della linea DATA corrente.
12. $0041—$0042 (=65—66)
    Puntatore all'indirizzo dei DATA.

Tutti questi puntatori sono già stati descritti in dettaglio nel 1° volume.

13. $0045—$0046 (=69—70)
    Nome della variabile BASIC corrente.
14. $0061—$0066 (=97—102)
    Accumulatore n.1 destinato a memorizzare le variabili in formato virgola mobile (utilizzato per il passaggio di parametri tramite l'istruzione USR).
15. $0069—$006E (=105—110)
    Accumulatore n. 2 destinato a memorizzare le variabili in formato virgola mobile.
16. $0073—$008A (=115—138)
    Sottoprogramma CHRGET, che serve per ricercare il byte successivo di un programma BASIC. Modificando questa routine è possibile estendere il BASIC del Commodore 64, aggiungendovi ad esempio delle istruzioni supplementari.
17. $0079 (=121)
    Punto di entrata della routine CHEGET che permette la ricerca dello stesso byte precedente in un programma BASIC.
18. $007A—$007B (=122—123)
    Quando l'interprete BASIC esegue un programma, esamina in successione i vari byte che lo compongono per estrarne le diverse istruzioni e variabili. Esiste perciò un puntatore che viene successivamente incrementato nel corso dell'esecuzione d'un programma BASIC, che è appunto collocato agli indirizzi $007A—$007B.

19. $0090 (=144)
    Byte di stato. Il valore assunto da questa locazione dopo un'operazione di ingresso/uscita viene restituito dall'istruzione BASIC STATUS.
20. $0098 (=152)
    In questa locazione troviamo il numero dei file binari aperti.
21. $0099 (=153)
    Qui è posto il codice della periferica di ingresso "per difetto" (0=tastiera).
22. $009A (=154)
    Qui troviamo il codice della periferica di uscita "per difetto" (3=schermo).
23. $00A0-$00A2 (=160—162)
    Queste locazioni permettono l'accesso al "timer" in tempo reale del Commodore 64. Esse forniscono infatti il numero di sessantesimi di secondo trascorsi dal momento dell'accensione della macchina.
24. $0087 (=183)
    In questa locazione è posta la lunghezza del nome del file corrente.
25. $00B8 (=184)
    In questa locazione troviamo il numero del file binario corrente.
26. $00B9 (=185)
    Fornisce il numero dell'indirizzo secondario (o comando) corrente.
27. $00BA (=186)
    Fornisce il numero della periferica correntemente in uso.
28. $00BB-$00BC (=187—188)
    Queste due locazioni contengono un puntatore che indica l'indirizzo dove è memorizzato il nome del file corrente.
29. $00C5 (=197)
    Questa locazione indica il tasto premuto al momento (0 indica che non è premuto alcun tasto).
30. $00C6 (=198)
    In questa locazione è contenuto il numero di caratteri presenti nel buffer della tastiera.
31. $0288 (=648)
    Come abbiamo visto nel 1° volume, questa locazione contiene l'indirizzo iniziale della memoria schermo, diviso per 256.
32. $0289 (=649)
    Contiene la dimensione del buffer di tastiera.
33. $028A (=650)
    Questa locazione determina quale tasto è dotato di una funzione AUTO-REPEAT. All'atto dell'accensione, essa contiene 0, e solo i tasti di comando del cursore ed il tasto dello spazio (SPACE) sono

dotati di auto-repeat. Caricando il valore 128 = $80 in questa locazione, tutti i tasti vengono a disporre della funzione AUTO-REPEAT.

34. $0291 (=657)
Il contenuto di questa locazione segnala se è abilitato oppure no lo SHIFT. Quando in essa c'è il valore 128 = $80 lo SHIFT è abilitato; mentre è disabilitato quando contiene 0.

35. $02A7-$02FF (=679-767)
Tutte queste locazioni non vengono utilizzate, e sono quindi liberamente disponibili, ad esempio per memorizzare un programma in linguaggio macchina.

36. $030C (=780)
Locazione utilizzata per memorizzare l'Accumulatore del 6510.

37. $030D (=781)
Come il precedente, per il registro indice X.

38. $030E (=782)
Come il precedente, ma per il registro indice Y.

39. $030F (=783)
Come il precedente, ma per il puntatore dello stack (registro SP).

40. $0310 (=784)
Questa locazione contiene l'istruzione di salto (JMP:$4C) utilizzata dall'istruzione USR.

41. $0311—$0312 (=785—786)
Queste locazioni contengono rispettivamente il byte basso e quello alto dell'indirizzo di inizio della routine in L.M. richiamata dall'istruzione USR.

42. $0313 (=787)
Non utilizzata.

43. $0314—$0315 (=788—789)
Contiene il vettore d'interrupt IRQ.

44. $0316—$0317 (=790—791)
Contiene il vettore dell'istruzione BREAK (interrupt programmato).

45. $0318—$0319 (=792—793)
Contiene il vettore d'interrupt non mascherabile NMI.

46. $0334—$033B (=820—827)
Locazioni non utilizzate.

47. $033C—$03FB (=828—1019)
Buffer di I/O per il lettore di cassette. Queste locazioni possono essere utilizzate altrimenti quando non viene impiegato il registratore a cassette.

48. $03FC—$03FF (=1020—1023)
Locazioni non utilizzate.

2.6 I sottoprogrammi del sistema

Abbiamo visto nel capitolo precedente una sintesi della configurazione hardware del Commodore 64. A questo punto è necessario, naturalmente senza addentrarci eccessivamente nei dettagli, vedere come funziona il sistema dal punto di vista "logico". Il sistema in questione, come abbiamo già avuto occasione di dire, è contenuto in due ROM di 8K byte ciascuna: una di esse contiene il BASIC (tabelle delle istruzioni e dei comandi, routine di calcolo in virgola mobile, ecc.); l'altra il sistema operativo. I compiti che quest'ultimo deve svolgere sono diversi:

— gestione dello schermo per il tramite del circuito di controllo 6566.
— gestione delle entrate/uscite (I/O) (tastiera, unità disco, registratore a cassette, stampante, ecc.)
— editare i testi nel corso dell'impostazione dei programmi BASIC.
— inizializzazione generale del sistema al momento dell'accensione.

Per darvi un'impressione del modo con cui interagiscono il sistema operativo ed il BASIC, facciamo un esempio. Sia il programma:

```
10 INPUT A
20 PRINT A
```

Quando l'interprete BASIC incontra il codice (singolo byte) dell'istruzione INPUT, esegue un salto ad una routine il cui compito è di attendere che venga premuto un tasto della tastiera.
Analogamente, quando l'interprete BASIC incontra l'istruzione (codice) PRINT, effettua un salto ad una routine che si incarica di visualizzare A sotto forma di una stringa di caratteri.
Queste varie routine di entrata/uscita, nonché numerose altre, sono accessibili all'utilizzatore grazie ad una tabella detta KERNAL. Questa tabella è composta da un gruppo di indirizzi ciascuno dei quali è relativo ad una determinata routine in L.M. Per fare un esempio, la routine che permette di sondare la tastiera viene lanciata da una chiamata di sottoprogramma che risiede all'indirizzo $FF9F (=65439).
Lo scopo del KERNAL è di poter godere dei vantaggi di compatibilità logica fra macchine diverse. Così, il VIC 20 ed il Commodore 64 utilizzano gli stessi indirizzi di salto, anche se poi le relative routine risultano differenti.
L'utilizzazione dei sottoprogrammi della ROM Monitor può essere realizzata seguendo un certo numero di stadi, che in genere sono questi:

— vengono richiamate alcune routine preparatorie (per esempio, l'inizializzazione d'un file binario);
— si caricano i registri di comunicazione (A, X, Y, per esempio) con i valori dei parametri da passare alla routine considerata;
— si effettua la chiamata della vera e propria routine con un'istruzione JSR;
— eventualmente, si provvede a gestire gli errori che si possono essere manifestati nel corso dell'esecuzione di tale routine.

Quando interviene un errore nell'esecuzione di una routine (ad esempio, se si tenta di caricare un file non esistente a partire da una cassetta o da un disco), il bit di riporto (CARRY) viene posto ad 1, e nell'accumulatore viene caricato un numero corrispondente al tipo di errore. La gestione di eventuali errori viene quindi effettuata prima in base ad un test sul bit di CARRY (istruzione BBC o BCS, secondo i casi), e poi con un test sul valore memorizzato nell'accumulatore.
I vari tipi di errori possibili sono quelli qui elencati:

Valore nell'Accumulatore	Tipo di errore
0	Routine fermata dalla pressione del tasto STOP
1	Numero di file aperti eccessivo
2	File già aperto in precedenza
3	File non aperto
4	File non reperito
5	Periferica non collegata
6	File non specificato in entrata
7	File non specificato in uscita
8	Manca il nome del file
9	Numero di periferica non ammesso
240	Allocazione errata del buffer associato al bus RS 232

Altri tipi di errori possono essere rivelati a livello del byte di stato (STATUS) posto all'indirizzo 144 = $0090. Per determinare questi errori oc-

corre impiegare la routine READST (Read Status, in inglese), che descriveremo più avanti.
Ed ora passiamo a descrivere le diverse routine messe a disposizione dei programmi in linguaggi macchina.

1. ACPTR

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFA5 = 65445	A	A,X	READST	Acquisizione dati dal bus seriale	TALK, TKSA

Questa routine viene utilizzata quando si desidera acquisire informazioni da una periferica collegata al bus seriale (ad esempio, il lettore di cassette). Il dato ricevuto viene memorizzato nell'accumulatore. Per poter venire usata, questa routine ha bisogno di essere "preparata" con l'ausilio di TALK e TKSA.
La routine TALK permette di ordinare ad una periferica di cominciare l'invio dei dati sul bus seriale. La routine TKSA, opzionale, permette di inviare un comando (od un indirizzo secondario) alla stessa periferica.
La gestione degli errori avviene per il tramite della routine READST (byte di stato). L'impiego di questa routine è molto semplice, e può venire illustrato da questo esempio:

```
C200 > JSR FFA5     ; acquisizione di un dato
C203 > STA FB       ; salvataggio in pagina zero
```

Il dato recuperato dall'accumulatore viene salvato in memoria in una locazione non utilizzata dal Monitor o dal BASIC (va bene l'indirizzo 251 = $FB). Osserviamo che il modo di funzionamento di questa routine è "a stretta di mano", o "handshaking" in inglese.

2. CHKIN

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFC6 = 65478	X	A,X	3,5,6	Apertura di un "canale" specificato in entrata	OPEN

Questa routine permette di definire in entrata un file binario che sia stato preventivamente aperto dalla routine OPEN. Naturalmente, se era già aperto non occorre ripetere l'apertura (la routine OPEN può essere seguita solo da un CLOSE).
La periferica corrispondente a tale file deve poter essere impiegabile come INPUT. Se tale routine viene usata con il bus seriale, il numero di file ed il comando (o indirizzo secondario) vengono alternativamente inviati alla periferica interessata.
Questa routine viene impiegata prima di CHRIN o GETIN, che vedremo più avanti. Nel caso in cui la periferica considerata è semplicemente la tastiera, questa routine non è necessaria, e nemmeno OPEN.
Il registro di comunicazione qui impiegato è il registro indice X, che contiene il numero del file desiderato.

Esempio:

```
C200 > LDX # 01       ; definisce in entrata il file binario n. 1
C202 > JSR FFC6
```

3. CHKOUT

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFC9= 65481	X	A, X	0,3,5,7 READST	Apertura di un "canale" specificato in uscita	OPEN

Questa routine funziona esattamente come quella precedente. Ovviamente, la periferica associata al numero di file deve poter essere impiegabile in uscita. Se questa routine deve funzionare con una periferica collegata al bus seriale, le devono essere fornite di seguito il numero di file binario (tramite il registro indice X) e l'indirizzo secondario eventuale.
Se l'unità di uscita è lo schermo, e non viene utilizzata altra periferica di uscita, non risulta necessario un preventivo impiego della routine OPEN.

Esempio:

```
C200 > LDX # 02        ; definisce in uscita il file logico n. 2
C202 > JSR FFC9
```

4. CHRIN

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFCF= 65487	A	A, X	0	Acquisizione di 1 carattere a partire da un "canale" specificato in entrata	OPEN, CHKIN

Questa routine viene usata congiuntamente alla routine CHKIN descritta in precedenza. Essa permette l'acquisizione di caratteri che provengono da una periferica specificata in entrata. La comunicazione avviene tramite l'accumulatore.
Se la tastiera costituisce l'unico dispositivo di entrata utilizzato, non c'è bisogno di utilizzare la routine CHKIN. I caratteri in tal caso sono memorizzati in un buffer utilizzato dall'istruzione INPUT, e possono venire acquisiti uno alla volta chiamando la routine CHRIN, sino al momento in cui viene trovato un "carriage return". A questo punto l'intero processo ricomincia da capo.

Esempio:

```
C200 > LDY # 00         ; inizializzazione del registro indice
C202 > JSR FFCF         ; acquisizione d'un carattere
C205 > STA 02A7,Y       ; salvataggio
C208 > INY              ; carattere successivo
C209 > CMP # 0D         ; l'ultimo carattere era un CR??
C20B > BNE FS           ; no, riprendi da capo
```

5. CHROUT

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFD2= 65490	A	A	0	Invio di un carattere a un canale specificato in uscita	CHKOUT, OPEN

Questa routine viene impiegata assieme alle routine CHKOUT ed OPEN. Essa permette di inviare un carattere ad un periferica il cui "canale" associato è specificato come uscita. Se l'unico dispositivo di uscita utilizzato è lo schermo, si possono omettere le routine CHKOUT ed OPEN.

Esempio:

```
C200 >LDA #46      ; invia il carattere F
C202 >JSR FFD2
```

6. CIOUT

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFA8= 65448	A	Nessuno	READST	Invio di un byte sul bus seriale	LISTEN, SECOND

Questa routine permette di inviare delle informazioni in forma di byte ad una periferica collegata sul bus seriale. È l'equivalente in uscita della routine ACPTR descritta prima. In tal senso, anch'essa funziona secondo il modo "handshaking". Per poter essere impiegata questa routine deve essere preparata con l'ausilio di LISTEN e di SECOND, se del caso. La routine LISTEN consente di ordinare ad una periferica di porsi in condizione di attesa di dati. La routine SECOND permette di inviare un comando od un indirizzo secondario ad una periferica funzionante in modo LISTEN.
La comunicazione avviene per il tramite dell'accumulatore.

Esempio:

```
C200 > LDA # 46      ; invio di una "F" sul bus seriale
C202 > JSR FFA8
```

7. CINT

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FF81 = 65409	Nessuno	A, X, Y	Nessuno	Inizializza l'editor d. schermo e il circuito di controllo tipo 6567	Nessuna

Questa routine permette di reinizializzare il circuito di controllo dello schermo, del tipo 6567, e di tornare al modo di visualizzazione normale (25 righe di 40 colonne). Essa compie quindi le medesime funzioni della pressione contemporanea dei tasti STOP e RESTORE (in regime BASIC).

Esempio:

```
C200 > JSR FF81
```

8. CLALL

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFE7 = 65511	Nessuno	A, X	Nessuno	Chiude tutti i file aperti	Nessuna

Questa routine permette di chiudere contemporaneamente tutti i "canali" precedentemente aperti da degli OPEN. Inoltre, essa chiama automaticamente la routine CLRCHN che vedremo più oltre.

Esempio:

```
C200 > JSR FFE7
```

9. CLOSE

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFC3 = 65475	A	A, X, Y	0, 240 READST	Chiudere un file binario	Nessuna

Contrariamente alla routine CLALL che chiude allo stesso tempo tutti i file binari aperti, questa routine permette di chiudere un file di numero assegnato. Questo numero va preliminarmente caricato nell'accumulatore.

Esempio:

```
C200 > LDA # 01        ; chiudere il file binario n. 1
C202 > JSR FFC3
```

10. CLRCHN

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFCC= 65484	Nessuno	A, X	Nessuno	Reinizializzazione dei "canali" di I/O	Nessuna

Questa routine permette di reinizializzare i "canali" di entrata/uscita ai loro valori "per difetto" (0 per la tastiera, e 3 per lo schermo), dopo il loro impiego. Se si omette questa routine, si può a volte provocare la ricezione delle stesse informazioni da parte di più periferiche diverse. (Nel caso che ciò fosse voluto, si farà naturalmente a meno di richiamare questa routine).

Esempio:

```
C200 > JSR FFCC
```

11. GETIN

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFE4= 65508	A	A, X, Y	READST	Acquisizione di 1 carattere	CHKIN, OPEN

Questa routine permette l'acquisizione di un carattere a partire da un canale specificato in entrata (e quindi inizializzato mediante le routine CHRIN ed OPEN).
Se viene utilizzata con la tastiera, un carattere presente nel buffer di ta-

stiera viene prelevato e trasferito all'accumulatore. Quando il buffer è vuoto, viene trasmesso il valore 0.
I caratteri vengono posti automaticamente nel buffer ogni volta che si preme un tasto (funzionamento mediante interrupt), sino a quando risulta pieno (10 caratteri). Dopo di che non viene accettato più alcun carattere sino a quando non ne viene prelevato almeno uno.

Esempio:

```
C200 > JSR FFE4      ; attendi un carattere
C203 > CMP # 00      ; è disponibile un carattere?
C205 > BEQ F9        ; no, attendi
```

12. IOBASE

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFF3 = 65523	X, Y	X, Y	nessuno	Definisce l'indirizzo di memoria delle periferiche di I/O	Nessuna

Questa routine permette di conservare la compatibilità logica fra VIC 20, Commodore 64 ed altri modelli futuri eventuali. Essa permette di conoscere l'indirizzo di inizio delle periferiche di entrata/uscita (I/O) del Commodore 64 (per esempio, della porta di I/O utente).
Il funzionamento di questa routine è molto semplice, ed è chiarito dall'esempio che segue:

```
C200 >JSR FFF3       ; richiamo della routine
C203 >STX FB         ; salvataggio del byte basso (=$00)
C204 >STY FC         ; salvataggio del byte alto (=$DC)
C206 > LDY #00       ;
C208 > LDA# 00
C20A >STA (FB), Y    ; carica 0 nella porta A del circuito CIA1
```

Dopo la chiamata di questa routine le locazioni $FB e $FC contengono i valori $00 e $0C, corrispondenti all'indirizzo $DC00 = 56320.
Mediante l'indirizzamento indiretto post-indicizzato è possibile accedere direttamente ai registri del circuito CIA1, ad esempio (qui la porta A).

13. IOINIT

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FF84 = 65412	Nessuno	A, X, Y	Nessuno	Inizializza tutte le periferiche e le routine di I/O	Nessuna

Questa routine permette di inizializzare tutte le periferiche di entrata/uscita e le relative routine associate. Essa viene eseguita ad ogni accensione del Commodore 64. Perciò, se scrivete dei programmi particolari che non fanno intervenire la ROM Monitor della vostra macchina, dovrete richiamare questa routine prima di effettuare qualsiasi tipo di comunicazione con l'esterno.

Esempio:

```
C200>JSR FF84
```

14. LISTEN

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFB1 = 65457	A	A	READST	Ordina ad una periferica collegata sul bus seriale di porsi in "ascolto"	Nessuna

Questa routine serve per ordinare ad una periferica collegata al bus seriale di porsi in modalità atta alla ricezione di dati. Questa si porrà quindi in condizione di attesa, ed accetterà poi le informazioni che le saranno inviate.
Il registro di comunicazione è l'accumulatore A, che deve preventivamente venire caricato col numero della periferica interessata (da 0 a 31).

Esempio:

```
C200>LDA#08     ; pone l'unità di lettura dischi in "ascolto"
C203>JSR FFB1
```

15. LOAD

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFD5 = 65493	A, X, Y	A, X, Y	0,4,5,8,9 READST	Caricamento di dati (file programma) da una periferica	SETLFS, SETNAM

Questa routine permette di introdurre direttamente nella memoria del Commodore 64 dei dati provenienti da una periferica esterna (lettore di cassette o di dischi). Se correttamente programmata, essa permette di ottenere gli stessi effetti del comando BASIC LOAD ma in maniera più generale: infatti con essa è possibile caricare pure programmi in linguaggio macchina.
Già il programma "Monitor in linguaggio macchina" di cui abbiamo già parlato dispone di comandi "LOAD", "SAVE" e "VERIFY" che funzionano con programmi scritti in L.M. Questi tre comandi impiegano la routine KERNAL "LOAD" e "SAVE" che descriveremo più avanti.
La routine LOAD funziona nel modo seguente:
Per prima cosa viene richiamata la routine SETLFS, che permette di creare un file binario. Poi viene chiamata la routine SETNAM che permette di assegnare un nome a questo file. Infine viene inizializzata la routine LOAD. Occorre innanzitutto caricare nell'accumulatore uno fra i valori 0 ed 1. Il valore 0 corrisponde ad un caricamento (LOAD), mentre il valore 1 corrisponde al VERIFY.
Se si richiede che il caricamento avvenga a partire dal medesimo indirizzo da cui il file è stato salvato, occorre inizializzare la routine SETLFS con un indirizzo secondario eguale ad 1. In tal caso, i registri indice X ed Y utilizzati come registri di comunicazione nella routine LOAD verranno pure inizializzati ponendovi il valore $FF.
Se invece si desidera "rilocare" il file (ossia caricarlo a partire da un indirizzo diverso da quello da cui era stato salvato), è necessario caricare nei registri X ed Y il byte basso ed alto del nuovo indirizzo di caricamento. L'indirizzo secondario inviato tramite la routine SETLSF sarà allora eguale a 0.
Nota: la routine LOAD non funziona con la tastiera, né con lo schermo, né con l'interfaccia RS 232: in pratica funziona esclusivamente con un lettore di dischi o di cassetta.

Esempio:

```
C200>LDA#00      ; caricamento senza rilocazione
C202>LDX#FF
C204>LDY#FF
C206>JSR FFD5

C200>LDA#01      ; verifica
C202>JSR FFD5
```

```
C200>LDA # 00      ; caricamento con rilocazione all'indirizzo
C202>LDX#20         $3020
C204>LDY#30
C206>JSR  FFD5
```

Va precisato che nel caso di un caricamento senza riallocazione (1° degli esempi precedenti), la routine LOAD rinvia, nei registri X ed Y, l'indirizzo dell'ultima locazione di memoria del Commodore 64 interessata dal caricamento.
Per conoscerne il valore, basterà impostare le linee seguenti:

```
C200>LDA#00        ; caricamento senza rilocazione
C202>LDX#FF
C204>LDY# FF
C206>JSR  FFD5
C209>STX FB        ; salva indirizzo di fine caricamento (byte
                    basso)
C20B>STY FC        ; byte alto
```

16. MEMBOT

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FF9C = 65436	X, Y	X, Y	Nessuno	Inizializza l'inizio della RAM disponibile	Nessuna

Questa routine permette di conoscere o di fissare l'indirizzo dell'inizio della memoria RAM utilizzata dall'interprete BASIC. All'atto della prima accensione della macchina, questo indirizzo vale $0800 = 2048, ma esso può venire modificato cambiando il valore del puntatore all'inizio della RAM disponibile. Questa routine provvede a svolgere automaticamente questo compito. Il registro indice X contiene il byte basso, ed il registro Y il byte alto di tale indirizzo.

Quando il bit di riporto (CARRY) è posto ad 1 prima della chiamata di questa routine, essa provvede a restituire il valore dell'indirizzo di inizio della RAM. Se tale bit si trova invece a 0, l'indirizzo di inizio della RAM verrà fissato eguale al contenuto dei registri X ed Y.

Esempio:

```
C200>SEC           ; lettura del puntatore all'inizio RAM
C201>JSR FF9C
C204>STX FB        ; salvataggio del byte basso
C206>STY FC        ; salvataggio del byte alto
C200>CLC           ; inizializzazione del puntatore all'inizio RAM
C201>LDX#00
C203>LDY#50
C205>JSR FF9C      ; indirizzo inizio RAM fissato a $5000 = 20480
```

17. MEMTOP

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FF99 = 65433	X, Y	X, Y	Nessuno	Inizializza la fine della RAM disponibile	Nessuna

Questa routine funziona esattamente allo stesso modo di quella precedente. Essa permette di conoscere o di fissare l'indirizzo dell'ultima locazione di memoria disponibile all'interprete BASIC.
Per un esempio di impiego, rifarsi alla precedente MEMBOT.

18. OPEN

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFC0= 65472	Nessuno	A, X, Y	1,2,4, 5,6,240 READST	Apertura di un file binario	SETLFS, SETNAM

Questa routine permette di "aprire" un file binario o "canale". Le caratteristiche di questo file devono essere fissate preventivamente con le routine SETLFS e SETNAM.

Esempio:

```
C200>JSR FFC0
```

19. PLOT

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFF0= 65520	X, Y	A, X, Y	Nessuno	Legge o fissa la posizione del cursore	Nessuna

Questa routine permette di conoscere o di fissare la posizione del cursore. Se il bit di riporto (CARRY) è posto ad 1 prima che venga chiamata questa routine, i registri indici vengono caricati col valore della posizione attuale del cursore. Il registro X contiene infatti il numero di riga (da 0 a 24), ed il registro Y il numero di colonna (da 0 a 39).

Se il bit del Carry viene invece posto a 0 prima di chiamare la routine, il cursore si disporrà nella posizione indicata dal contenuto dei due registri indice X ed Y.

Esempio:

```
C200>SEC          ; lettura della posizione del cursore
C201>JSR FFF0
C204>STX FB       ; salvataggio del numero di riga
C206>STY FC       ; salvataggio del numero di colonna

C200>CLS          ; fissa la posizione del cursore
C201>LDX#08       ; alla riga n. 8
C203>LDY#12       ; ed alla colonna n. 18
C205>JSR FFF0
```

20. RAMTAS

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FF87 = 65415	A, X, Y	A, X, Y	Nessuno	Test della RAM	Nessuna

Questa routine permette di effettuare un test della RAM del Commodore 64. Essa inizializza i puntatori all'inizio ed alla fine del BASIC, e l'inizio della memoria dello schermo, e pone a zero le locazioni di memoria dall'indirizzo $0000 a $0101 nonché $0200 a $03FF.
Essa viene utilizzata dalla ROM Monitor all'atto di ogni accensione della macchina.

21. RDTIM

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFDE = 65502	A, X, Y	A, X, Y	Nessuno	Legge l'orologio in tempo reale	Nessuna

Questa routine consente di leggere l'orologio (timer) in tempo reale del Commodore 64. Il risultato è in sessantesimi di secondo, trascorsi dall'istante in cui la macchina è stata accesa. Questo numero viene trasmesso tramite l'accumulatore (byte alto), il registro indice X (byte intermedio) ed il registro indice Y (byte basso).
Il tempo trascorso si ricava quindi dalla formula

$$T = 65536*(A) + 256*(X) + (Y) \text{ in sessantesimi di secondo}$$

Esempio:

```
C200>JSR FFDE      ; legge il timer in tempo reale
C203>STA FB
C205>STX FC
C207>STY FD
```

22. READST

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFB7 = 65463	A	A	Nessuno	Legge il byte di stato (STATUS)	Nessuna

Questa routine permette di restituire nell'accumulatore il valore del byte di stato relativo agli organi di entrata/uscita (I/O). Questo byte di stato viene modificato da ogni operazione di ingresso ed uscita dati.
La tabella seguente mostra i valori diversi che vengono assunti dal byte di stato in funzione degli errori riscontrati durante una di queste operazioni (sono riportati solo i tipi di errori principali).

Periferica	Bit n°	Valore	Tipo di errore
Cassetta	4	16	errore di lettura
	5	32	errore nella "cheksum"
	6	64	termine del file
	7	−128	termine del nastro
Portaseriale	0	1	ritardo trascorso in scrittura
	1	2	ritardo trascorso in lettura
	6	64	termine della linea
	7	−128	periferica non collegata

Esempio:

```
C200>JSR FFB7      ; legge il byte di stato
C203>AND#40        ; termine del file?
C205>BNE*1         ; sì, passare alla gestione di tale errore
```

Quando viene rivelata la fine del file, il bit n° 6 del byte di stato viene posizionato ad 1. Si ha allora il salto al Label *1, che rappresenta l'indirizzo dove comincia una routine per la gestione di tale tipo di errore (per esempio, con la visualizzazione del messaggio "Fine del file").

23. RESTOR

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FF8A= 65418	. Nessuno	A, X, Y	Nessuno	Ristabilisce i valori per difetto dei vettori usati dal BASIC e dal Monitor	Nessuna

Questa routine permette di ristabilire i valori "per difetto" dei vettori utilizzati dal BASIC e dal Monitor, in particolare i vettori per la gestione degli interrupt.

Esempio:

```
C200>JSR FF8A
```

24. SAVE

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFD8= 65496	A, X, Y	A, X, Y	5, 8, 9 READST	Salvataggio di una parte della memoria	SETLFS, SETNAM

Questa routine consente di salvare una parte della memoria su di una periferica, quale un lettore di cassette o di dischi.
Essa richiede la chiamata preliminare delle routine SETLFS e SETNAM incaricate di definire il file binario utilizzato. Come nel caso della routine LOAD la routine SAVE non funziona con la tastiera, né con l'interfaccia RS 232, né con lo schermo.
Prima di chiamare la routine SAVE nell'accumulatore va caricato il valore che indica l'indirizzo di un puntatore in pagina zero.
Il contenuto di A fornisce il byte basso dell'indirizzo dove comincia la parte di memoria da salvare. All'indirizzo seguente ((A) + 1) si trova il byte alto.
Inoltre, i registri X ed Y devono venire caricati con il byte basso ed alto dell'indirizzo del termine della parte di memoria da salvare.

Esempio:

```
C200>LDA#00     ; inizio parte da salvare = $C000
C202>STA FB     ; inizializza il puntatore in pagina zero (byte
                  basso)
C204>LDA#C0
C206>STA FC     ; byte alto
C208>LDA FB
C20A>LDX#00     ; termine della parte da salvare = $C100
C20C>LDY#C1
C2DE>JSR FFD8
```

25. SCNKEY

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FF9F = 65439	Nessuno	A, X, Y	Nessuno	Sondaggio della tastiera	IOINIT

Questa routine permette di sondare la tastiera per verificare se è premu-

to uno dei suoi tasti. In tal caso, il relativo codice ASCII viene posto in un buffer della tastiera, che può contenere, come abbiamo già detto, un massimo di 10 caratteri.
Questa routine viene chiamata soltanto se il vettore IRQ non è stato modificato.

Esempio:

```
C200>JSR FF9F      permette di sondare la tastiera

C203>JSR FFE4      permettono di leggere entro l'accumula-
C206>CMP#00        tore il codice ASCII del tasto premuto
C208>BEQ F6

C20A>JSR FFD2      permette di visualizzare il carattere
```

26. SCREEN

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFED = 65517	X, Y	X, Y	Nessuno	Fornisce il formato dello schermo	Nessuna

Questa routine permette di conoscere il formato attuale dello schermo (in generale formato da 25 righe di 40 colonne).

Esempio:

```
C200 JSR FFED
```

A questo punto il registro X contiene il numero di colonne visualizzate ed il registro Y quello delle righe.

27. SECOND

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FF93= 65427	A	A	READST	Invio di un comando ad una periferica	LISTEN

Questa routine permette di inviare un comando od un indirizzo secondario ad una periferica che si trova in condizione di "ascolto" (e pertanto inizializzata con la routine LISTEN). Se si vuole inviare un comando sul bus seriale, occorre preventivamente porre ad 1 i bit 5 e 6 ("OR" logico con $60).
Il comando va caricato nell'accumulatore prima di chiamare la routine.

Esempio:

```
C200>LDA#0F        ; invio del comando 15=$0F
C202>JSR FF93
```

28. SETLFS

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFBA= 65466	A, X, Y	Nessuno	Nessuno	Inizializzazione di un file binario	Nessuna

Questa routine permette di fissare le caratteristiche d'un file binario, vale a dire il suo numero, il numero della periferica corrispondente, nonché il comando (o indirizzo secondario) da inviare a questa.
I numeri delle periferiche sono, vogliamo ricordarlo, 0 per la tastiera, 1 per il lettore di cassette, 2 per l'interfaccia RS 232, 3 per lo schermo, 4 o 5 per la stampante, ed 8 per l'unità disco.
La procedura da seguire è la seguente:

— si carica nell'accumulatore il numero del file binario (da 1 a 255)
— si carica nel registro indice X il numero della periferica considerata
— si carica nel registro indice Y il comando. Se non deve essere inviato alcun comando, tale registro conterrà il valore 255 = $FF.
— si chiama infine la routine SETLFS.

Esempio:

```
C200>LDA#01      ; numero di file = 1
C202>LDX#08      ; periferica = lettore di dischi
C204>LDY#07      ; comando = 15
C206>JSR FFBA
```

29. SETMSG

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FF90 = 65424	A	A	Nessuno	Controllo dei messaggi sullo schermo	Nessuna

Questa routine permette il controllo della visualizzazione di messaggi di errore e di controllo.

Esempio:

DEVICE NOT PRESENT è un messaggio di errore
PRESS PLAY ON TAPE è un messaggio di controllo

A seconda del valore presente nell'accumulatore potranno venire visualizzati un messaggio di errore od un messaggio di controllo.
Se il bit n. 6 di A è posto ad 1, viene abilitata la visualizzazione di messaggi di controllo. Se il bit n. 7 di A è ad 1, è abilitata la visualizzazione di messaggi di errore.

Esempi:

```
C200>LDA#40     ; abilita un messaggio di controllo
C202>JSR FF90

C200>LDA#80     ; abilita un messaggio di errore
C202>JSR FF90

C200>LDA#00     ; interdice l'emissione di messaggi dei due
C202>JSR FF90     tipi
```

30. SETNAM

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFBD= 65469	A, X, Y	Nessuno	Nessuno	Assegna un nome ad un file	Nessuna

Questa routine permette di assegnare un nome ad un file inizializzato mediante la routine SETLFS, per utilizzarlo assieme alle routine OPEN, SAVE o LOAD. Prima occorre caricare nell'accumulatore la lunghezza del nome adottato. Questo sarà memorizzato sotto forma

d'una sequenza di caratteri ASCII. I registri indice X ed Y conterranno rispettivamente il byte basso ed alto dell'indirizzo di memoria del primo carattere del nome.

Esempio:

```
C200> LDA# 04    ; lunghezza del nome = 4 caratteri
C202> LDX# A7    ; il nome è memorizzato a partire dall'indirizzo
                   $02A7
C204> LDY #02
C206> JSR #FFBD
```

Per il nome di file "TEST" le locazioni di memoria d'indirizzo da S02A7 a S02AA conterranno i valori seguenti:

($02A7) = $54
($02A8) = $45
($02A9) = $53
($02AA) = $54

31. SETTIM

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFDB = 65499	A, X, Y	Nessuno	Nessuno	Inizializza il timer in tempo reale	Nessuna

Così come la routine RDTIM permette di leggere il timer in tempo reale del Commodore 64, la routine SETTIM permette di aggiornarne l'ora. I due funzionamenti sono assai simili, salvo che qui i registri A, X ed Y devono venire caricati con il numero di sessantesimi di secondo richiesti, prima di chiamare SETTIM.

Notiamo che l'orologio in tempo reale del Commodore 64 permette di misurare il tempo sino a 24 ore, corrispondenti a 5.184.000 sessantesimi di secondo; dopo di che esso si rimette automaticamente a zero.

Esempio: Per disporre il timer all'ora 14h28m35s (3.126.900 sessantesimi di secondo, si imposterà il programmino seguente:

```
C200>LDA#2F    ; byte alto
C202>LDX#86    ; byte intermedio
C204>LDY#74    ; byte basso
C206>JSR FFDB
```

32. SETTMO

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFA2 = 65442	A	Nessuno	Nessuno	Posiziona il "flag" di abilitazione di una temporizzazione con il bus IEEE	Nessuna

Questa routine viene impiegata solo nel caso in cui risulti collegata al Commodore 64 una cartuccia IEEE. Essa permette di ottenere una temporizzazione (ritardo) di 64 millisecondi nel caso in cui si impieghi una periferica con standard IEEE. Se questa periferica non risponde al Commodore 64 quando questo le invia un dato, entro questo lasso di tempo, interviene un errore e la macchina sospende la sua attesa. Quando il bit 7 dell'accumulatore è a 0 è abilitata la temporizzazione. Invece, quando tale bit 7 vale 1, la temporizzazione è interdetta, ed il Commodore 64 attenderà sino a che la periferica fornisca una risposta.

Esempio:

```
C200>LDA#80      ; temporizzazione interdetta
C202>JSR FFA2

C200>LDA#00      ; temporizzazione abilitata
C202>JSR FFA2
```

33. STOP

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFE1 = 65505	A	A, X	Nessuno	Verifica se è stato premuto il tasto STOP	Nessuna

Questa routine permette di controllare se è stato premuto il tasto STOP al momento della chiamata della routine UDTIM che vedremo più oltre, e che ha il compito di aggiornare il timer in tempo reale.
Se così è il caso, la chiamata della routine STOP posizionerà ad 1 il bit Z del registro di stato P del 6510. Inoltre, i diversi "canali" verranno reinizializzati al loro valore "per difetto".
Nel caso contrario, ossia se il tasto non era premuto, il bit Z è a 0, e l'accumulatore contiene un byte che rappresenta l'ultima linea verificata sulla tastiera.

Esempio:

```
C200>JSR FFEA    ; chiama la routine UDTIM
C203>JSR FFE1    ; attende che venga premuto il tasto STOP
C206>BNE F8
```

34. TALK

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFB4 = 65460	A	A	READST	Ordina ad una periferica posta sul bus seriale di mettersi in modo "invio di dati"	Nessuna

Questa routine funziona allo stesso modo della routine LISTEN, salvo che la periferica interessata si mette qui ad emettere dei dati. Preliminarmente si sarà caricato nell'accumulatore il numero della periferica.

Esempio:

```
C200>LDA#08       ; periferica n. 8 in modo "emissione dati"
C202>JSR FFB4
```

35. TKSA

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FF96 = 65430	A	A	READST	Invia un comando ad una periferica in modo "emissione"	TALK

Questa routine permette di inviare un comando ad una periferica che sia stata preventivamente posta in modo "emissione dati" dalla routine TALK. Occorre preliminarmente caricare nell'accumulatore il numero del comando da inviare.

Esempio:

```
C200>LDA#08      ; periferica n. 8 in modo emissione dati
C202>JSR FFB4
C205>LDA#0F      ; invio del comando "15" alla data periferica
C207>JSR FF96
```

36. UDTIM

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFEA = 65514	Nessuno	A, X	Nessuno	Aggiornamento del timer in tempo reale	Nessuna

Questa routine aggiorna il tempo segnato dal timer in tempo reale. Normalmente essa viene chiamata automaticamente ogni sessantesimo di secondo tramite interrupt. Comunque, chiamando questa routine è possibile modificare "manualmente" il tempo segnato dal timer in tempo reale, in particolare quando si usa una routine di gestione degli interrupt diversa da quella presente nel Commodore 64.

Esempio:

```
C200>JSR FFEA
```

37. UNLSN

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFAE= 65454	Nessuno	A	READST	Ordina a tutte le periferi-che poste sul bus seriale di fermare la ricezio-ne di dati	Nessuna

Questa routine permette di ordinare a tutte le periferiche collegate sul bus seriale, che abbiano in precedenza ricevuto un ordine LISTEN, di interrompere la ricezione dei dati provenienti dal Commodore 64.

Esempio:

```
C200>JSR FFAE
```

38. UNTLK

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FFAB= 65451	Nessuno	A	READST	Ordina a tutte le periferi-che poste sul bus seriale di arrestare l'invio di dati	Nessuna

Questa routine funziona esattamente nella stessa maniera delle precedente UNLSN, però questa volta si riferisce alle periferiche che sono state preventivamente inizializzate con la routine TALK.

Esempio:

```
C200>JSR FFAB
```

39. VECTOR

Indirizzo	Registri di comunicazione	Registri influenzati	Errori	Funzione	Routine preliminari
$FF8D = 65421	X, Y	A, X, Y	Nessuno	Gestione dei vettori posti nella RAM	Nessuna

Questa routine serve per gestire i vettori (o indirizzi di salto) utilizzati dal sistema operativo e collocati entro la RAM.
Quando il bit di riporto (CARRY) è posto ad 1 quando viene chiamata questa routine, i valori dei vettori vengono memorizzati a partire da un indirizzo che è determinato dal contenuto dei registri X ed Y (rispettivamente contenenti il byte basso ed alto dell'indirizzo).
Quando invece tale bit di riporto è posto a 0 quando viene chiamata questa routine, la tabella dei vettori utente (eventualmente modificati da questo), che si trova memorizzata a partire dall'indirizzo puntato dal contenuto dei registri X ed Y, viene trasferita nella zona riservata alla memorizzazione di tali vettori.

Esempio:

```
C200>LDX#00     ; trasferisce la tabella dei vettori nella zona
                  che parte dall'indirizzo $C000
C202>LDY#C0
C204>SEC
C205>JSR FF8D
```

```
C200>LDX#00      ; carica la tabella dei vettori tramite la par-
C202>LDY#C0        te di RAM che inizia a $C000
C204>CLC
C205>JSR FF8D
```

E così siamo finalmente alla fine di questo lungo paragrafo che si è occupato della routine KERNAL.
Affronteremo ora un nuovo capitolo, che confidiamo vi metterà in grado di realizzare della grafica in alta risoluzione ad una velocità nettamente superiore a quella raggiunta nel 1° volume, dove tutti i programi erano scritti in BASIC.

3
L'animazione nella grafica

Nel 1° libro di quest'opera abbiamo descritto in dettaglio la procedura da seguire per realizzare la visualizzazione ad alta risoluzione sullo schermo. Come esempio di applicazione avevamo inoltre ivi descritto il tracciamento di una curva.
In questo capitolo torneremo su tale esempio, e vedremo come sia possibile prima di tutto tracciare un punto, quindi un'intera curva sullo schermo, e ciò con una velocità nettamente superiore a quella possibile mediante un programma scritto completamente in BASIC.

3.1 Introduzione

La realizzazione della visualizzazione in alta risoluzione si ottiene attraverso un certo numero di fasi successive che vogliamo brevemente ricordare.

1. Selezione della pagina di memoria di 16 K byte indirizzata dal circuito di controllo dello schermo. La cosa si realizza tramite il circuito di controllo degli I/O CIA2. Per maggiori dettagli su questo punto e su alcuni altri che incontreremo più avanti, vi preghiamo di riferirvi al capitolo 3 del 1° volume.

2. Selezione della collocazione della memoria schermo, che diventa ora la memoria colore per la visualizzazione.

3. Selezione di un generatore di caratterio di 8K byte, che conterrà le informazioni relative allo stato di "acceso" o "spento" dei vari punti sullo schermo.

4. Selezione del modo grafico ad alta risoluzione.

5. Inizializzazione della visualizzazione.
Per questo, basta riempire tutte le locazioni di memoria che compongono il generatore di caratteri con il valore 0. Inoltre, si può programmare uno stesso colore per ciascun punto riempiendo la memoria colore (in precedenza, memoria schermo) con il codice del colore desiderato (ad es. 1 per il colore bianco).

6. Determinazione delle coordinate X e Y del punto considerato.
A partire da questi due valori è possibile ricavare la posizione esatta del bit corrispondente nel generatore di caratteri.

7. Determinazione del numero di riga dove sarà posto il punto (da 0 a 24).

8. Determinazione del numero di colonna dove si troverà il punto (da 0 a 39).

9. Determinazione della riga interessata nella matrice di punti di dimensioni 8 × 8 posta all'ordinata RIGA ed all'ascissa COLONNA, determinate in precedenza.

10. Determinazione del byte in memoria corrispondente.

11. Determinazione del bit interessato nel byte suddetto.

12. "Accensione" o "spegnimento" del punto corrispondente.

Nel paragrafo che segue considereremo le diverse routine che si impiegano per realizzare le varie operazioni ora descritte.

3.2 Visualizzazione di un punto

Tutte le routine che verranno descritte più oltre sono state descritte e verificate con l'ausilio del monitor in linguaggio macchina che abbiamo messo a punto per il Commodore 64, e di cui abbiamo già parlato. Questo programma occupa dello spazio memoria, e perciò abbiamo dovuto fare alcune scelte per quanto riguarda ad es., la pagina di memoria indirizzata dal circuito di controllo dello schermo. Ben inteso, voi potete modificarne la collocazione, badando però attentamente che nè il generatore di caratteri nè la memoria colore vengano interferite da un altro programma residente in memoria.

1. Selezione della pagina di 16 K byte indirizzata dal circuito di controllo dello schermo

La pagina n. 0 (quella che viene selezionata quando viene accesa la macchina) non può venire impiegata nel nostro caso, perché il nostro monitor in L.M. occupa troppo spazio in memoria. Questo è il motivo per cui noi abbiamo scelto di indirizzare con il circuito di controllo dello schermo la pagina n. 1, posta fra gli indirizzi

16384 = $4000 e 32767 = $7FFF

In BASIC, la cosa si realizza così:

```
POKE 56578, PEEK(56578) OR 3
POKE 56576, (PEEK(56576) AND 252) OR 2
```

In Assembler si potrebbe invece scrivere

```
C000>LDA DD02
C003>ORA # 03
C005>STA DD02
C008>LDA DD00
C00B>AND # FC
C00D>ORA # 02
C00F>STA DD00
```

Questa routine svolge esattamente le stesse funzioni delle due linee di BASIC precedenti ma, ovviamente, con molta maggiore velocità.

2. Selezione della collocazione della memoria schermo

Come sappiamo, la pagina di memoria indirizzata dal circuito di controllo dello schermo è posta fra gli indirizzi 16384 e 32767. Vedremo più avanti che il generatore di caratteri occupa 8 K byte posti fra gli indirizzi 24576 e 32767. Pertanto, una buona collocazione per la memoria schermo risulta essere quella fra gli indirizzi 23552 e 24575.
In BASIC si scriverà allora:

```
POKE 53272, (PEEK(53272) AND 15) OR 112
```

Ed in Assembler:

```
C012>LDA D018
C015>AND # 0F
C017>ORA # 70
C019>STA D018
```

3. Selezione d'un generatore di caratteri di 8 K byte

Questo dovrà essere situato nella seconda metà della pagina di 16 K byte indirizzata dal circuito di controllo dello schermo. E dunque verrà posto fra gli indirizzi 24576 e 32767.
Per ottenere tale collocazione basta posizionare i bit n. 3, 2 ed 1 del registro d'indirizzo $D018 = 53272 rispettivamente ad 1, 0, 0, ossia caricarvi $08.
In BASIC si scriverà allora:

```
POKE 53272, (PEEK(53272) AND 240) OR 8
```

Ed in Assembler:

```
C01C>LDA D018
C01F>AND # F0
C021>ORA # 08
C023>STA D018
```

4. Selezione del modo grafico ad alta risoluzione

Basta a questo scopo porre ad 1 il bit 5 del registro di indirizzo 53265 = $D011 del circuito di controllo dello schermo.
In BASIC scriveremo;

```
POKE 53265, PEEK(53265) OR 32
```

Ed in Assembler:

```
C026> LDA D011
C029>ORA # 20
C02B>STA D011
```

5. Inizializzazione della visualizzazione

Dobbiamo riempire tutte le locazioni comprese fra gli indirizzi 24576 e 32767 con il valore 0 (=inizializzazione del generatore di caratteri). In BASIC si scriverà:

```
10 FOR I=24576 TO 32767
20 POKE I 0
30 NEXT I
```

Se avete fatto "girare" il programma di tracciamento di una curva fornitovi nel 1° volume, vi sarete accorti del tempo che richiede questa inizialízzazione.
Con la routine che segue, questa operazione diventa quasi istantanea:

```
C02E > LDA # 60      ; puntatore all'indirizzo $600=24576
C030 > STA FC
C032 > LDA # 00
C034 > STA FB
C036 > LDY # 00      ; inizializzazione contatore dei cicli
C038 > STA(FB),Y     ; carica 0 in memoria
C03A > DEY           ; pagina di 256 byte terminata?
C03B > BNE FB        ; no, torna da capo
C03D > INC FC        ; sì, pagina seguente
C03F > LDA FC
C041 > CMP # 80      ; è l'ultima pagina?
C043 > BCC ED        ; no, riprendi da capo
```

Agli indirizzi $FB e $FC si trovano il byte basso ed alto dell'indirizzo in cui si deve caricare il valore 0. Il riempimento si fa per pagine di 256 byte, mediante indirizzamento indiretto post-indicizzato. Il procedimento viene replicato fino a che si raggiunge l'ultima pagina. Nello stesso modo con cui abbiamo riempito le locazioni corrispondenti al generatore di caratteri con il valore 0, possiamo programmare un colore uniforme per lo schermo. Se scegliamo ad esempio il colore bianco, ci basterà caricare il valore $01 nelle locazioni comprese fra 23552=$C000 e 24575=$5FFF.
In BASIC scriveremo:

```
10 FOR I = 23552 TO 24575
20 POKE I, 1
30 NEXT I
```

Ed in Assembler:

```
C045>LDA # 5C      ; puntatore all'indirizzo $5C00 = 23552
C047>STA FC
C049>LDA # 00
C04B>STA FB
C04D>LDA # 01      ; colore bianco
C04F>LDY # 00
C051>STA (FB),Y
C053>DEY           ; pagina di 256 byte terminata?
C054>BNE FB        ; no, torna da capo
C056>INC FC        ; sì, pagina seguente
C058>LDA FC
C05A>CMP # 60      ; è l'ultima pagina?
C05C>BCC EF        ; no, riprendi da capo
CO5E>RTS
```

Il funzionamento di questa routine è perfettamente analogo a quella che ci è servita per inizializzare il generatore di caratteri.
Si deve notare che la totalità d'una pagina di 1 K byte (1024 byte) viene caricata con valori uno. La memoria schermo occupa però solo 1000 byte: quelli che restano possono servire come puntatori per gli SPRITE. Se impiegherete comunque questo ultimo modo di visualizzazione, dovrete stare attenti a non distruggere questi puntatori se essi si troveranno agli stessi indirizzi della memoria schermo selezionata.

6. Determinazione delle coordinate X ed Y del punto considerato

Nel caso di tracciamento di una curva, tali coordinate possono venir calcolate mediante un programma BASIC (come descritto nel 1° volume). Esse dovranno poi essere passate come parametri ad una routine in L.M. che si incarica di visualizzare il punto.
Le coordinate X ed Y possono variare, rispettivamente, fra i valori 0 e 319, ovvero 0 e 199. Il passaggio del valore di X richiede quindi 2 byte (il valore può superare 255), mentre per il passaggio del valore di Y ne basta uno solo. Si potranno ad esempio adottare i registri di indirizzo

$FC = 252, $FD = 253 e $FE = 254

7. Determinazione del numero di riga dove si troverà il punto

Si vuol realizzare la seguente operazione

```
RIGA = INT(Y/8)
```

In Assembler questa operazione si può eseguire assai semplicemente mediante tre "spostamenti verso destra" di bit (istruzione LSR).
Si potrà così scrivere:

```
C05F>LDA # FC      ; valore di Y
C061>LSR   A
C062>LSR   A
C063>LSR   A       ; valore di INT (Y/8)
C064>STA   FB      ; salvataggio del risultato = RIGA
```

Il risultato RIGA viene salvato all'indirizzo $FB = 251.

8. Determinazione del numero di colonna dove si troverà il punto

Qui vogliamo realizzare il calcolo

```
COLONNA = INT (X/8)
```

Prima di operare come nel caso del calcolo del numero di riga, va precisato che il valore COLONNA dovrà successivamente venire moltiplicato per 8 nel calcolo del byte di memoria corrispondente.
Infatti

```
BYTE = 24576 + 320*RIGA + 8*COLONNA + LC
```

Noi dovremo quindi calcolare il valore 8*INT(X/8).
Questa operazione si può eseguire "mascherando" i tre bit inferiori del byte basso che rappresenta il valore di X.
Dato che X è memorizzato nelle locazioni di indirizzo $FD (byte basso) e $FE (byte alto), sarà sufficiente scrivere;

```
C066>LDA # FD      ;
C068>STA C0DD      ; salvataggio del byte basso di X
C06B>AND #F8       ; 8*COLONNA
C060>STA FD        ; salvataggio in $FD e $FE
```

Il risultato 8 * INT (X/8) viene memorizzato in $FD e $FE, mentre il byte basso di X viene salvato in $C0DD per un calcolo successivo.

9. Determinazione della riga nella matrice di dimensioni 8×8 posta all'ascissa COLONNA ed all'ordinata RIGA

Vogliamo realizzare il calcolo di

LC = Y AND 7

Poi tale risultato va sommato al valore 8*COLONNA trovato precedentemente.
Potremo scrivere

```
C06F>LDA FC         ; valore di Y
C071>AND # 07       ; Y AND 7
C073>CLC
C074>ADC FD         ; somma 8*COLONNA
C076>STA FD
C078>LDA FE
C07A>ADC # 00
C07C>STA FE         ; 8*COLONNA + LC in $FD e $FE
```

10. Determinazione del byte di memoria interessato

Dobbiamo determinare ora il valore di

BYTE = 24576 + 320*RIGA + 8*COLONNA + LC

Questa risulta l'operazione senz'altro più complessa, perché si tratta di operare su "parole" di 16 bit e di realizzare una moltiplicazione per 320.
Il programma relativo è il seguente:

```
C07E>LDA # 00       ; byte alto = 0
C080>STA FC
C082>LDA FB
C084>LDX # 06       ; calcolo di 64*RIGA
C086>ASL A
```

```
C087 >ROL FC
C089 >DEX
C08A>BNE FA
C08C>STA FB
C08E >STA C0DE      ; salva 64*RIGA
C091 >LDA FC
C093 >STA C0DF
C096 >LDA C0DE      ; calcolo di 256*RIGA
C099 >ASL A
C09A>ROL C0DF
C02D>ASL A
C09E>ROL C0DF
C0A1>CLC            ; calcolo di 320*RIGA
C0A2>ADC FB
C0A4 >STA FB
C0A6>LDA C0DF
C0A9>ADC FC
C0AB>STA FC
C0AD>CLC            ; calcolo di 320*RIGA+8*COLONNA+LC
C0AE>LDA FB
C0B0>ADC FD
C0B2>STA FB
C0B4>LDA FC
C0B6>ADC FE
C0B8>ADC # 60       ; calcolo di BYTE
C08A>STA FC         ; risultato in $FB e $FC
```

Questo programma funziona nel seguente modo:
Prima di tutto si calcola 64*RIGA e poi 256*RIGA. Questi due valori vengono poi sommati assieme, e così si ottiene 320*RIGA. Il risultato viene salvato in $FB e $FC. Ad esso si somma poi il valore di B*COLONNA+LC, salvato in precedenza in $FD e $FE, ed infine si somma $6000 ossia 24576.
Il risultato finale, che rappresenta l'indirizzo del byte considerato, viene memorizzato agli indirizzi $FB e $FC (byte basso in $FB e byte alto in $FC).

11. Determinazione del bit interessato

Si vuole eseguire il calcolo

```
BIT = 7 —(X AND 7)
```

Qui serve nuovamente quindi il byte basso del valore di X (salvato in precedenza all'indirizzo $C0DD).
Potremo scrivere allo scopo la seguente routine:

```
C0BC>LDA C0DD    ; valore del byte basso di X
C0BF>AND # 07
C0C1>STA FD      ; X AND 7 in $FD
C0C3>LDA # 07
C0C5>SEC
C066 >SBC FD     ; 7—(X AND 7)
C0C8>STA FD      ; salva in $FD
```

Il valore del bit è dunque salvato in $FD.

12. "Accensione" o "spegnimento" del punto

Perché il punto venga "illuminato", si deve realizzare l'equivalente dell'istruzione BASIC

```
POKE BYTE, PEEK (BYTE) OR 2 ↑ BIT
```

La cosa si può realizzare con la routine seguente:

```
C0CA >LDA # 01     ;
C0CC >LDX FD
C0CE >BEQ 04       ; BIT = 0: passo dello spostamento
C0D0 >ASL A        ; sposta A = $01 in BIT volte
C0D1 >DEX
C0D2 >BNE FC       ; 2 ↑ BIT
C0C4 >LDY # 00
C0D6 >ORA (FB),Y   ; PEEK (BYTE) OR 2 ↑ BIT
C0D8 >STA (FB),Y   ; POKE BYTE, PEEK (BYTE) OR 2 ↑ BIT
C0DA >RTS
```

Il calcolo di 2 ↑ BIT viene effettuato mediante un numero di spostamenti verso sinistra (ASL) del valore $01 pari a BIT.
Allo stesso modo, per "spegnere" il punto, l'istruzione BASIC necessaria è

```
POKE BYTE, PEEK (BYTE) AND (255-2 ↑ BIT)
```

La routine Assembler equivalente è:

```
C0CA > LDA # 01
C0CC > LDX FD
C0CE > BEQ 04
C0D0 > ASL A
C0D1 > DEX
C0D2 > BNE FC        ; 2 ↑ BIT
C0C4 > LDY # 00
C0D6 > EOR # FF      ; 255—2 ↑ BIT
C0D8 > AND (FB),Y    ; PEEK (BYTE) AND (255—2 ↑ BIT)
C0DA > STA (FB),Y    ; POKE BYTE, (PEEK(BYTE) AND (255—2 ↑
                       BIT))
C0DC > RTS
```

Questa routine ha un funzionamento simile alla precedente, e non abbisogna quindi di particolari chiarimenti.
Concatenando le diverse routine descritte in questa sezione, sarà dunque possibile illuminare (o spegnere) un dato punto dello schermo.
Come avrete potuto probabilmente notare, il numero di istruzioni del 6510 necessarie è rilevante, però il risultato è senz'altro spettacolare.

3.3 Tracciamento di una curva in alta risoluzione

Ci proponiamo ora di descrivere sullo schermo la stessa curva data negli esempi del 1° volume di quest'opera, ma questa volta utilizzando le routine in linguaggio macchina viste sopra.
Ovviamente, una parte del programma sarà ancora in BASIC, e cioè quella destinata ad effettuare la ricerca dei valori massimo e minimo ed al calcolo delle coordinate di ciascun punto.
Il programma BASIC che adotteremo è il seguente:

```
10 REM PROGRAMM PER TRACCIARE UNA CURVA
20 DEF FNA(X) = SIN(X)/X
30 INPUT "INTERVALLO?"; X1, X2
40 PA = (X1 — X2)/320
50 MI = 1.7 E 38 : MA = — 1.7 E 38
60 FOR X=0 TO 319
70 X3 = X1 + X * PA
80 IF X3=0 THEN Y=1:GOTO 100
```

```
90 Y = FNA (X3)
100 IF Y  MI THEN MI = Y
110 IF Y  MA THEN MA = Y
120 NEXT
130 B = MA * 199 / (MA — MI):A = 199 / (MI — MA)
140 SYS(49152)
150 FOR X = 0 TO 319
160 X3 = X1 + X * PA
170 IF X3=0 THEN Y = INT(A+B): GOTO 190
180 Y = INT(B + A * FNA(X3))
190 POKE 252,Y : XH = INT(X/256) : POKE 253,X—256 * XH: POKE
    254, XH
200 SYS(49247):NEXT X
210 GOTO 210
```

L'istruzione SYS (49152) richiama la routine di inizializzazione della visualizzazione ad alta risoluzione (indirizzi da $C000 a $C05E).
L'istruzione SYS (49247) corrisponde invece alla visualizzazione vera e propria di un punto (indirizzi da $C05F a $C0DA). Il passaggio dei valori dei parametri X ed Y avviene tramite le locazioni di memoria di indirizzi 252=$FC, 253=$FD e 254=$FE.
L'impiego di queste due routine (quella di inizializzazione e quella di visualizzazione) non è limitata ovviamente al tracciamento di una curva sullo schermo. Esse costituiscono comunque la base di tutte le visualizzazioni di questo tipo, e potranno da voi essere utilmente impiegate anche nei vostri programmi BASIC od Assembler.

3.4 L'utilizzo degli SPRITE

Nel 1° volume abbiamo visto che la programmazione dei propri SPRITE corrisponde, pressapoco, a programmare alcuni registri del circuito di controllo dello schermo nonché una tabella di dati che rappresenta il movimento degli "sprite".
In questa sezione riprendiamo per prima cosa l'esempio del gioco PAC-MAN descritto nel 1° volume, fornendo una routine in L.M. che permette di programmare questi piccoli "mostri", per poi descrivere alcune tecniche particolari che ci permetteranno di usare questi SPRITE in giochi di movimento.

3.4.1 Programmazione di PAC-MAN

I valori da caricare in memoria, che rappresentano PAC-MAN, sono ricordati qui sotto:

BYTE 0 = 0	BYTE 1 = 3	BYTE 2 = 255
BYTE 3 = 0	BYTE 4 = 63	BYTE 5 = 254
BYTE 6 = 1	BYTE 7 = 255	BYTE 8 = 252
BYTE 9 = 7	BYTE 10 = 252	BYTE 11 = 248
BYTE 12 = 31	BYTE 13 = 252	BYTE 14 = 240
BYTE 15 = 63	BYTE 16 = 255	BYTE 17 = 224
BYTE 18 = 127	BYTE 19 = 255	BYTE 20 = 192
BYTE 21 = 127	BYTE 22 = 255	BYTE 23 = 128
BYTE 24 = 255	BYTE 25 = 255	BYTE 26 = 0
BYTE 27 = 255	BYTE 28 = 254	BYTE 29 = 0
BYTE 30 = 255	BYTE 31 = 252	BYTE 32 = 0
BYTE 33 = 255	BYTE 34 = 254	BYTE 35 = 0
BYTE 36 = 255	BYTE 37 = 255	BYTE 38 = 0
BYTE 39 = 127	BYTE 40 = 255	BYTE 41 = 128
BYTE 42 = 127	BYTE 43 = 255	BYTE 44 = 192
BYTE 45 = 63	BYTE 46 = 255	BYTE 47 = 224
BYTE 48 = 31	BYTE 49 = 255	BYTE 50 = 240
BYTE 51 = 7	BYTE 52 = 255	BYTE 53 = 248
BYTE 54 = 1	BYTE 55 = 255	BYTE 56 = 252
BYTE 57 = 0	BYTE 58 = 63	BYTE 59 = 254
BYTE 60 = 0	BYTE 61 = 3	BYTE 62 = 255

Il programma in L.M. comprenderà innanzitutto questa tabella di dati, ed un certo numero di istruzioni che hanno il compito di memorizzarli in una ideonea collocazione di memoria (zona di definizione degli SPRITE), e di definire le dimensioni, il colore e la posizione degli SPRITE stessi.
Scriveremo quindi:

```
C000 > DFB   00
C001 > DFB   03
C002 > DFB   FF
C003 > DFB   00
C004 > DFB   3F
C005 > DFB   FE
C006 > DFB   01
```

```
C007 > DFB    FF
C008 > DFB    FC
C009 > DFB    07
C00A > DFB    FC
C00B > DFB    F8
C00C > DFB    1F
C00D > DFB    FC
C00E > DFB    FD
C00F > DFB    3F
C010 > DFB    FF
C011 > DFB    E0
C012 > DFB    7F
C013 > DFB    FF
C014 > DFB    C0
C015 > DFB    7F
C016 > DFB    FF
C017 > DFB    80
C018 > DFB    FF
C019 > DFB    FF
C01A > DFB    00
C01B > DFB    FF
C01C > DFB    FE
C01D > DFB    00
C01E > DFB    FF
C01F > DFB    FC
C020 > DFB    00
C021 > DFB    FF
C022 > DFB    FE
C023 > DFB    00
C024 > DFB    FF
C025 > DFB    FF
C026 > DFB    00
C027 > DFB    7F
C028 > DFB    FF
C029 > DFB    80
C02A > DFB    7F
C02B > DFB    FF
C02C > DFB    C0
C02D > DFB    3F
C02E > DFB    FF
C02F > DFB    E0
```

```
C030 > DFB   1F
C031 > DFB   FF
C032 > DFB   F0
C033 > DFB   07
C034 > DFB   FF
C035 > DFB   F8
C036 > DFB   01
C037 > DFB   FF
C038 > DFB   FC
C039 > DFB   00
C03A > DFB   3F
C03B > DFB   FE
C03C > DFB   00
C03D > DFB   03
C03E > DFB   FF
```

e quindi:

```
C03F >LDA   # 0B   ; puntatore allo SPRITE n. 1: indirizzo 704
C041 >STA   07F8     = $02C0
C044 >LDX   # 3E
C046 >LDA   C000,X; carica la definizione dello SPRITE in me-
C049 >STA   02C0,X  moria
C04C >DEX
C04D >BNE   F7
C051 >LDA   # 01   ; visualizza lo SPRITE n. 0
C053 >STA   D015
C056 >STA   D027   ; colore dello SPRITE: bianco
C059 >LDA   # 06   ; colore dello sfondo: blu
C05B >STA   D021
C05E >LDA   # 80   ; posizione dello SPRITE
C060 >STA   D000   ; asse X (8 bit del byte basso)
C063 >STA   D001   ; asse Y
C066 >LDA   # 00   ; asse X (byte alto)
C068 >STA   D010
C06B >RTS
```

Il funzionamento di questo programma è del tutto simile a quello del programma BASIC che abbiamo fornito nel 1° volume. Esso verrà lanciato, nel nostro caso, a partire dall'indirizzo $C03F.
Nell'esempio ora dato la posizione dello SPRITE è fissa. Naturalmente si può fare in modo che esso si sposti, come abbiamo visto fare nel 1° li-

bro, ma questa volta renderemo il movimento assai più veloce.
Per cominciare, vogliamo fare attraversare al nostro PAC-MAN lo schermo da sinistra verso destra.
La routine BASIC che ci permetteva di farlo era, lo ricordiamo:

```
170 FOR I=0 TO 344
180 POKE 53264, INT(I/256)
190 POKE 53248, I—256 * INT(I/256)
200 NEXT I
```

Ora vogliamo realizzarla in Assembler.
Impostate dunque la seguente routine:

```
C03F>LDA   # 0B
C041>STA   07F8
C044>LDX   #3E
C046>LDA   C000,X
C049>STA   02C0,X
C04C>DEX
C04D>BNE   F7
C04F>LDA   # 01
C051>SSTA  D015
C054>STA   D027
C057>LDA   # 06
C059>STA   D021
C05C>LDA   # 80
C05E>STA   D001
C061>LDA   # 00   ; SPRITE sul lato sinistro dello schermo:
                    X=0
C063>STA   FB
C065>STA   FC
C067>LDA   FB
C069>STA   D000
C06C>LDA   FC
C06E>STA   D010
C071>INC   FB     ; X=X+1
C073>LDA   FC
C075>CMP   # 01
C077>BEQ   09
C079>LDA   FB
C07B>BNE   EC
C07D>INC   FC
```

```
C07F>JMP   C069
C082>LDA   FB
C084>CMP   # 59    ; X=345?
C086>BNE   E1      ; no, ricomincia
C088>RTS           ; si, fine
```

Le prime linee di questa routine sono simili a quella precedente.
Provvedete a lanciarne l'esecuzione dall'indirizzo $C03F, ed osservate attentamente lo schermo: non vedrete niente, o quasi niente, salvo forse un breve lampo chiaro!
E tuttavia lo SPRITE ha attraversato completamente lo schermo da sinistra a destra, ma ad una velocità incredibile!
Anche per un gioco di movimento che voglia essere veloce, occorre dunque provvedere a rallentarne un poco la velocità, o mediante una routine di ritardo, o facendo eseguire contemporaneamente al microprocessore degli altri compiti. Per prima cosa esaminiamo il caso di un ciclo di ritardo (ciclo di attesa).
A partire dall'indirizzo $C061 il programma va così modificato:

```
C061>LDA   # 00
C063>STA   FB
C065>STA   FC
C067>LDA   FB
C069>LDX   # FF   ; ciclo di ritardo
C06B>DEX
C06C>BNE   FD
C06E>STA   D000
C071>LDA   FC
C073>STA   D010
C076>INC   FB
C078>LDA   FC
C07A>CMP   # 01
C07C>BEQ   # 09
C07E>LDA   FB
C080>BNE   E7
C082>INC   FC
C084>JMP   C069
C087>LDA   FB
C089>CMP   # 59
C08B>BNE   DC
C08D>RTS
```

Ed ora provate a lanciare il programma (sempre dall'indirizzo $C03F): dovreste vedere lo SPRITE attraversare lo schermo da sinistra verso destra, questa volta ad una velocità più lenta che nel caso precedente, ma sempre molto più rapida che nel caso del programma scritto in BASIC come quello descritto nel 1° volume.
Ora supponiamo di voler visualizzare contemporaneamente sullo schermo diversi SPRITE, generare dei rumori, ecc.
È senz'altro possibile memorizzare le definizioni di più SPRITE e farli muovere l'uno appresso all'altro.
L'algoritmo per fare ciò potrebbe essere questo:

In ogni caso bisognerà inserire un ciclo di ritardo, a meno che nel frattempo non facciate svolgere altre operazioni come visualizzare qualcosa in alta risoluzione, fare dei calcoli, ecc.
Lo spostamento degli SPRITE lungo l'asse Y si effettua in modo del tutto simile. Potete riferirvi al 1° volume per quanto riguarda le routine necessarie per una tale operazione.

Ovviamente, il movimento di uno SPRITE può avvenire secondo una traiettoria qualsiasi: basterà semplicemente modificare simultaneamente sia l'ascissa X che l'ordinata Y.
Passeremo ora ad esaminare qualche tecnica speciale, impiegabile ad esempio per dei giochi di movimento.

3.5 Qualche tecnica di animazione

3.5.1 La grafica ad alta risoluzione

Abbiamo visto che quando si utilizza questo metodo di visualizzazione il generatore di caratteri deve venire memorizzato entro gli ultimi 8K byte dell'area di memoria indirizzata dal circuito di controllo dello schermo. La programmazione di ciascun punto di tale generatore di caratteri richiede del tempo, anche quando la si effettua in linguaggio macchina.
Per certi giochi di movimento che prevedono degli sfondi "simili" (ossia poco diversi fra loro se non in alcuni particolari), può essere interessante memorizzare ciascuno di tali disegni entro un generatore di caratteri diverso, e commutare questi ultimi (ossia effettuare il passaggio da uno ad un altro) mediante un semplice cambio di pagina di memoria. La cosa si può fare grazie al registro d'indirizzo 56576, come abbiamo già visto nel 1° e nel 2° volume.
Allo stesso modo, se si desidera passare "istantaneamente" da un "paesaggio" ad un altro, si potrà ottenere lo scopo con le operazioni seguenti:

— visualizzare il primo "paesaggio": generatore di caratteri n° 1.
— contemporaneamente, definizione punto per punto del secondo "paesaggio": generatore di caratteri n° 2 (senza che il 1° disegno venga influenzato in alcun modo).
— commutazione della pagina di memoria per il circuito di controllo dello schermo: viene quindi visualizzato il secondo "paesaggio", mentre il primo può venire, se richiesto, salvato o modificato a vostro piacimento, sempre senza alcuna influenza sul disegno che compare al momento sullo schermo.

3.5.2 Gli SPRITE

Abbiamo visto come a ciascuno SPRITE sia associato un puntatore posto negli ultimo 8 byte dello spazio di 1K byte associato alla memoria dello schermo.
Supponiamo che si voglia disporre di alcuni SPRITE la cui forma deve variare leggermente in funzione del tempo.
Per ottenere questo, basterà definire altrettanti puntatori di SPRITE (e quindi altrettante memorie schermo diverse) quanti sono richiesti. A ciascuno di tali puntatori verrà associato uno spazio di memoria di 63 byte per la definizione dello SPRITE corrispondente. Commutando l'indirizzo di inizio della memoria dello schermo (come descritto nel 1° volume, tramite il registro d'indirizzo 53272), sarà possibile passare da una definizione di SPRITE ad un'altra, e quindi eventualmente modificare la sua andatura.
Questa tecnica si può usare vantaggiosamente nel caso dei "mostri" tipo PAC-MAN la cui bocca si apre e si chiude.
Le tecniche che abbiamo menzionato qui sopra sono destinate a darvi alcuni concetti validi per realizzare i vostri giochi animati. Comunque, dovrete sempre prestare buona attenzione alla posizione di memoria dove sono allocati i generatori di caratteri, le memorie schermo, gli SPRITE, ecc., nonché la pagina indirizzata dal circuito di controllo dello schermo. Tutto questo richiede una buona conoscenza della mappa di memoria del Commodore 64 e dei circuiti di controllo dello schermo. Vi rimandiamo quindi ad un nuovo studio dei capitoli dedicati a questo argomento.

4
Le entrate e le uscite (INPUT/OUTPUT)

4.1 Generalità

Come tutti i microcomputer oggi disponibili sul mercato, il Commodore 64 è costruito tutt'attorno ad un microprocessore (che nel nostro caso è il 6510), che svolge tutte le funzioni di calcolo necessarie per un dato programma in memoria, sia che questa sia memoria di sola lettura (ROM) o memoria "attiva" (RAM). È evidente comunque che un microprocessore non può limitarsi ad operare entro uno spazio circoscritto, ma deve anche comunicare col mondo esterno.
Chiameremo pertanto "Entrate" (o ingressi, o INPUT con un termine inglese molto diffuso in informatica) tutte le forme di acquisizione di informazioni per il microprocessore a partire da un'unità o dispositivo esterno (tastiera, lettore di cassette, unità disco, ecc.); ed "Uscite" (od OUTPUT) tutte le forme di trasmissione di informazioni da parte del microprocessore rivolte ad un dispositivo esterno (schermo, stampante, registratore a cassette, unità disco; ecc.). Nel caso del Commodore 64 le informazini passano attraverso circuiti particolari specializzati nella gestione dei dati di entrata/uscita (I/O), chiamati CIA (Complex Interface Adapter), qui del tipo 6526.
Questo circuto ha delle prestazioni eccellenti, risulta programmabile, ed ha il compito di scaricare il microprocessore da varie operazioni che, se esso non esistesse, mobiliterebbero il 6510 impedendogli in pratica di svolgere altre funzioni nello stesso tempo.
Fra le funzioni realizzate dal 6526 (o meglio, dai 6526, perché sul Commodore 64 ce ne sono due) possiamo ricordare:

— due porte di I/O parallele A e B, ad 8 bit, programmabili, che consentono di realizzare funzioni diverse che dettaglieremo in seguito

— due timer di 16 bit ciascuno, programmabili, con funzioni multiple
— un registro a spostamento parallelo → serie e serie → parallelo
— un "orologio" che può fornire l'ora (in ore, minuti, secondi e decimi di secondo), dotato anche di un allarme programmabile.

Prima però di entrare nei dettagli circa il funzionamento di questo circuito, ci sembra preferibile iniziare mostrando un esempio di applicazione, in modo da farvene ben comprendere l'utilità.

4.2 Realizzazione d'un'entrata/uscita semplificata

Lo scopo che ci prefiggiamo è, molto semplicemente, di verificare se un interruttore risulta aperto o chiuso, e di comandare in corrispondenza l'accensione o lo spegnimento di un LED (diodo elettroluminescente) ad opera del 6526.
Il piccolo circuito che abbiamo costruito a questo scopo è estremamente semplice, non può provocare alcun danno al Commodore 64 quando lo si collega, e soprattutto non richiede alcuna conoscenza preventiva dell'elettronica.
Inoltre, esso può condurre ad applicazioni anche diverse, quali:

— un sistema di allarme per un'abitazione (il Commodore 64 può controllare ad esempio se porte e finestre sono tutte chiuse, e attivare un circuito di allarme in caso contrario).
— un sistema di controllo automatico per elettrodomestici od impianti di riscaldamento centrale (per questo tipo di applicazioni risultano assai utili da un lato il timer in tempo reale, e dall'altro i due timer integrati entro i circuiti tipo 6526).
— realizzazione di giochi di luce; ecc.

L'elenco appena descritto non è da intendersi come limitativo, e voi potrete dare libero sfogo alla vostra immaginazione sull'argomento.
Prima di descrivere le diverse operazioni da effettuare per realizzare questo tipo di entrata/uscita, vogliamo descrivere i diversi connettori e prese accessibili all'utilizzatore.
Il Commodore 64 è munito di due porte di controllo e di una porta utente. Le porte di controllo sono in particolare destinate al collegamento delle espansioni tipo PADDLE, JOYSTICK, penna ottica (Light Pen), ma possono anche venire impiegate per applicazioni I/0 da parte dell'utilizzatore.

Invece, la porta utente è stata studiata in modo particolare per questo genere di operazioni, ed in particolare viene impiegata per il sintetizzatore di voce tipo VOTRAX o per il MODEM distribuiti dalla Commodore.

4.2.1 La porta utente

Questa porta è costituita da un connettore direttamente riportato sul circuito stampato che comprende tutti i componenti del Commodore 64.
Per accedervi, basterà che acquistiate un connettore femmina compatibile (2 × 12 spinotti con passo 3.96 mm), perché certamente non è il caso di effettuare connessioni saldate sul circuito stampato del vostro computer. Infatti in primo luogo potrebbe essere compromessa l'eventuale garanzia, ed in secondo luogo potreste danneggiare la vostra macchina, cosa senz'altro deprecabile.
I piedini del connettore sono numerati da 1 a 12 sulla faccia superiore, e da A ad N su quella inferiore. Eccone le corrispondenze:

Piedino	Funzione
1 = massa	massa generale del computer
2 = +5V	alimentazione (positivo)
3 = $\overline{\text{RESET}}$	se si pone a 0 questa entrata, si provoca la reinizializzazione del Commodore 64: la memoria non viene cancellata, ma si perdono i programmi BASIC a causa di modifiche ai relativi puntatori
4 = CNT1	circuito CIA1: entrata esterna utilzzata dai timer
5 = SP1	circuito CIA1: uscita seriale
6 = CNT2	circuito CIA2: entrata esterna utilizzata dai timer
7 = SP2	circuito CIA2: uscita seriale
8 = PC2	circuito CIA2: linea per la modalità "handshaking"
9 = SERIALATN	linea ATN del bus seriale
10 = 9VAC + 11 = 9VAC −	collegamento alla tensione alternata proveniente dal secondario del trasformatore di alimentazione (max 50 mA)
12 = massa	massa generale
A = massa	massa generale
B = FLAG2	circuito CIA2: collegamento utile all'"handshaking"
C = PBO	circuito CIA2: linea n° 0 della porta B
D = PB1	circuito CIA2: linea n° 1 della porta B
E = PB2	circuito CIA2: linea n° 2 della porta B
F = PB3	circuito CIA2: linea n° 3 della porta B
G =	non collegato
H = PB4	circuito CIA2: linea n° 4 della porta B
I	non collegato
J = PB5	circuito CIA2: linea n° 5 della porta B
K = PB6	circuito CIA2: linea n° 6 della porta B
L = PB7	circuito CIA2: linea n° 7 della porta B
M = PA2	circuito CIA2: linea n° 2 della porta A
N = massa	massa generale

4.2.2 Le porte di controllo

Queste sono due, situate sul lato destro della macchina. Utilizzano dei connettori a 9 spinotti. Per accedervi, basterà munirvi di connettori femmina corrispondenti.

Il vostro "Manuale d'uso" vi fornisce le indicazioni sul ruolo svolto da ciascun piedino di queste porte (per il collegamento ai joystick, ai paddle, alla penna ottica = Light Pen, ecc.)
Se desiderate collegarvi altri tipi di circuiti, vi occorre conoscere quale tipo di segnale corrisponde esattamente a ciascun piedino.
Queste indicazioni sono fornite dalle tabelle seguenti:

PORTA 1:

Piedino	Funzione
1	circuito CIA1: linea n° 0 della porta B
2	circuito CIA1: linea n° 1 della porta B
3	circuito CIA1: linea n° 2 della porta B
4	circuito CIA1: linea n° 3 della porta B
5	potenziometro "Y": circuito controllo suono
6	circuito CIA1: linea n° 4 della porta B + Light Pen
7	+5V
8	massa
9	potenziometro "X": circuito controllo suono

PORTA 2:

Piedino	Funzione
1	circuito CIA1: linea n° 0 della porta A
2	circuito CIA1: linea n° 1 della porta A
3	circuito CIA1: linea n° 2 della porta A
4	circuito CIA1: linea n° 3 della porta A
5	potenziometro "Y": circuito controllo suono
6	circuito CIA1: linea n° 4 della porta A
7	+5V
8	massa
9	potenziometro "X": circuito controllo suono

Ora però torniamo alla costruzione della nostra porta di I/0 semplificata. Esamineremo passo a passo le varie operazioni da compiere:

— È evidente che l'operazione che consiste nel leggere lo stato di interruttore è un'entrata: il 6526 possiede quindi due registri, DDRA e DDRB (Data Direction Register A e B) che permettono di configurare ciascuna delle porte parallele A e B come entrata o come uscita.

Sia DDRA che DDRB, così come pure le porte A e B, sono viste dal microprocessore come semplici locazioni di memoria (ad 8 bit).
Si può quindi scrivere entro DDRA e DDRB. E, secondo i casi, si può leggere o scrivere nelle porte A e B.
La messa ad 1 di certi bit dei registri DDRA e DDRB configura come uscite i bit corrispondenti della porta relativa.
Inversamente, la messa a zero di certi bit dei registri DDRA e DDRB configura come entrate i bit corrispondenti della relativa porta.

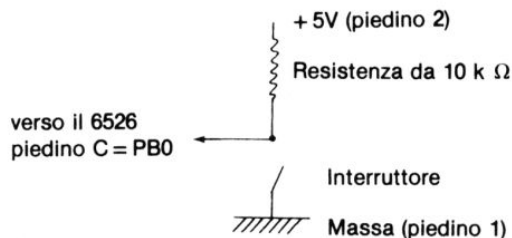
Esempio:

bit	7							0
DDRB	1	1	1	0	0	1	0	1

I bit 0, 2, 5, 6 e 7 sono configurati come uscita
I bit 1, 3 e 4 sono configurati come entrata

Nel seguito ci interesseremo alla porta utente, ed in particolare alle linee da PB0 a PB7 (piedini C, D, E, F, H, J, K, L del connettore) del circuito CIA2. Gli indirizzi dei vari registri utilizzati al momento sono

Porta B: $DD01 = 56577
DDRB : $DD03 = 56579

Per poter leggere lo stato d'un interruttore, basterà montare il circuitino che segue:

Il materiale necessario è il seguente:

— un interruttore unipolare
— una resistenza di circa 10 kohm
— un connettore da 2 × 12 piedini (passo 3,96 mm)

Prima di porre mano al vostro saldatore, leggete attentamente i suggerimenti che seguono, a scanso di dover ricorrere al servizio di assistenza tecnica del vostro Commodore 64!

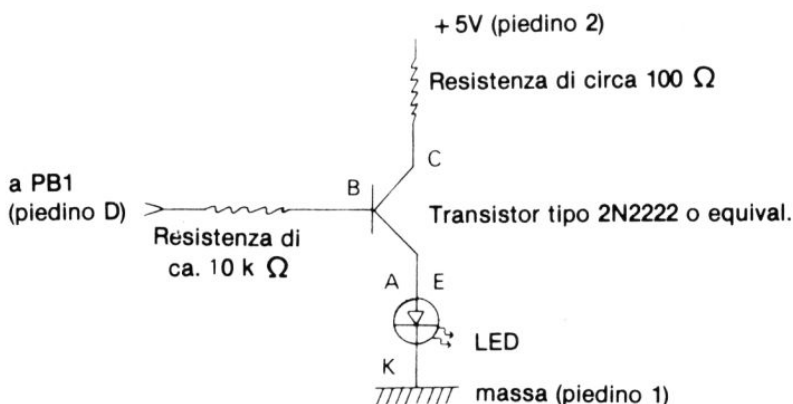
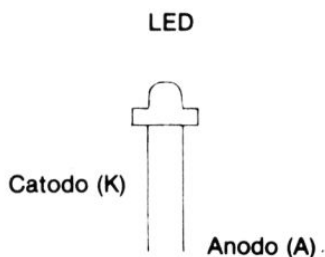
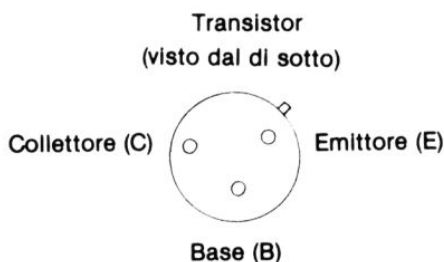
— Come abbiamo detto prima, non effettuate mai alcuna saldatura direttamente sul circuito stampato del vostro microcomputer: utilizzate dei connettori, anche se costano più cari!
— Non effettuate mai alcuna saldatura sui piedini di un connettore quando questo è inserito e la macchina è sotto tensione.
— Non inserite mai un connettore sulla relativa presa con la macchina sotto tensione: rischiate di provocare dei corti circuiti.
— Prima di dare tensione definitivamente, controllate più volte il vostro circuito.

Detto questo, non abbiate altri timori: il vostro microcomputer non corre assolutamente alcun rischio, se seguite le precauzioni indicate!
Esso inoltre risulta protetto da un fusibile, e la sua alimentazione dispone di una protezione elettronica, per cui in genere vi basterà cautelarvi munendovi di qualche fusibile di ricambio.
Una volta realizzato e collegato il circuitino elementare descritto sopra, battete sulla tastiera le due linee:

```
10 POKE 56579,0
20 PRINT PEEK (56577) AND 1
```

Se l'interruttore è aperto, dovete ottenere il valore 1; se invece risulta chiuso, dovete leggere 0 sullo schermo.
Naturalmente, potete accedere alla porta I/0 anche tramite il linguaggio macchina.
Il relativo programma si scrive:

```
C000>LDA # 00
C002>STA DD03      ; configura la porta B come entrata
C005>LDA DD01      ; legge lo stato dell'interruttore
C008>AND # 01      ; salva PB0
```

Ed ora vediamo come si può realizzare un'uscita.
Il circuito che dovete costruire risulta in questo caso un tantino più complicato, perché richiede anche un transistor. Ma, rassicuratevi, queste "bestioline" sono, contrariamente a certe idee diffuse, molto "robuste" (a condizione, naturalmente, di non trattarle davvero brutalmente).
Ecco il circuito in questione:

Le piedinature del transistor e del LED sono le seguenti:

Questo circuito, come il precedente, potrà comodamente venire costruito sopra una piastra tipo "Veroboard", dotata di molti fori praticati su diverse piste conduttrici di rame.
Ovviamente, andrà bene anche qualsiasi altro tipo di supporto conveniente.

Dopo di che, battete le linee seguenti:

```
10 POKE 56579,2           ; PB0 in entrata, PB1 in uscita
20 POKE 56577,0           ; LED spento
30 FOR I=1 TO 20:NEXT
40 POKE 56577,2           ; LED acceso
50 FOR I=1 TO 20:NEXT I
60 GOTO 20
```

Se date un RUN, vedrete lampeggiare il diodo luminescente (LED). In Assembler, il programma corrispondente si scrive:

```
C000>LDA # 02    ; PB0 in entrata, PB1 in uscita
C002>STA DD03
C005>LDA # 00    ; LED spento
C007>STA DD01

C000>LDA # 02
C002>STA DD03
C005>STA DD01    ; LED acceso
```

Abbiamo così visto come è possibile realizzare delle porte INPUT/OUTPUT semplificate con l'ausilio del 6526. Naturalmente, potete aumentare il numero di entrate/uscite, per realizzare ad esempio un vero e proprio circuito di controllo per usi domestici.
Non dimenticate, se desiderate comandare degli apparecchi che funzionano alimentati dalla rete, di prevedere delle idonee interfacce (ad esempio un accoppiatore ottico + un triac od un relais di potenza adatta).
Qui non descriveremo la realizzazione di circuiti di questo tipo, che trascenderebbero gli scopi di quest'opera.
Passiamo invece a descrivere in modo dettagliato quell'eccezionale circuito che è il 6526.

4.3 Descrizione dettagliata del 6526

Proprio come succede per i circuiti di controllo dello schermo e del suono, descritti nel 1° volume, il 6526 appare al microprocessore come un assieme di locazioni di memoria nelle quali esso può, secondo i casi, leggere o scrivere.

Nel 6526 ce ne sono 16, ed il loro ruolo è descritto nella tabella che segue:

Registro	*Indirizzo CIA1*	*Indirizzo CIA1*	*Funzione*
PRA	56320 = $DC00	56576 = $DD00	Porta A
PRB	56321 = $DC01	56577 = $DD01	Porta B
DDRA	56322 = $DC02	56578 = $DD02	Data Direction Register A
DDRB	56323 = $DC03	56579 = $DD03	Data Direction Register B
TA L	56324 = $DC04	56580 = $DD04	Timer A: byte basso
TA H	56325 = $DC05	56581 = $DD05	Timer A: byte alto
TB L	56326 = $DC06	56582 = $DD06	Timer B: byte basso
TB H	56327 = $DC07	56583 = $DD07	Timer B: byte alto
TOD 10e	56328 = $DC08	56584 = $DD08	Orologio: decimi di secondo
TOD SEC	56329 = $DC09	56585 = $DD09	Orologio: secondi .
TOD MIN	56330 = $DC0A	56586 = $DD0A	Orologio: minuti
TOD HRS	56331 = $DC0B	56587 = $DD0B	Orologio: ore
SDR	56332 = $DC0C	56588 = $DD0C	Registro per le comunicazioni seriali
ICR	56333 = $DC0D	56589 = $DD0D	Registro di controllo degli interrupt
CRA	56334 = $DC0E	56590 = $DD0E	Registro di controllo A
CRB	56335 = $DC0F	56591 = $DD0F	Registro di controllo B

I nomi assegnati ai registri (come ad es. TOD, ICR, ...) corrispondono a quelli indicati nel "data sheet" del costruttore, che nel caso è americano; per cui corrispondono in genere alle iniziali di termini inglesi. Esamineremo distintamente le seguenti funzioni:

— le entrate-uscite parallele
— i timer
— l'"orologio"
— le comunicazioni in modo seriale
— gli interrupt (interruzioni)

4.3.1 Le Entrate/Uscite parallele

Abbiamo visto, nel caso delle entrate/uscite semplificate che abbiamo descritto in precedenza, si dovevano inizialmente caricare nei registri

DDRA e DDRB i valori richiesti, per poter configurare le porte A e B come uscite. Ma il 6526 è in grado di svolgere funzioni molto più complesse, in particolare per quanto riguarda le comunicazioni con le periferiche. Esistono allo scopo due piedini, detti PC e FLAG, che facilitano tale compito. Questi segnali sono disponibili sulla porta utente, ma solo per il circuito CIA2.
Il piedino PC funziona come uscita, mentre il piedino FLAG funziona come entrata. Il loro funzionamento è il seguente:
Il segnale che esce da PC normalmente si trova a livello alto (circa +5V). Quando viene effettuata un'operazione di lettura o scrittura sulla porta B, questo segnale passa a livello basso per la durata di due cicli di temporizzazione.
Esso può venire impiegato in due casi:

— per indicare ad una periferica che il dato che essa ha appena inviato è stato accettato
— per indicare ad una periferica che si sta per inviarle un dato.

Queste due operazioni fanno parte della procedura detta in modo assai espressivo, di "handshaking", ossia "con stretta di mano".
Il trasferimento dei dati avviene sotto forma di "parola" (byte) di 8 bit, ma è pure possibile, in teoria, un funzionamento con i bit delle porte A e B. Purtroppo, come sappiamo, sulla porta utente del Commodore 64 è presente solo la porta B, ragion per cui non tratteremo questo caso.
Il piedino FLAG in entrata è sensibile ad una transizione (fianco) negativo, ossia al passaggio da +5V a 0V. Ogni volta che una tale transizione è rivelata dal 6526, il bit d'interrupt FLAG è posto ad 1. Questo bit è collocato entro il registro di controllo degli interrupt (bit 4), di cui parleremo più oltre.
Quando una periferica invia al 6526 un segnale di questo tipo, gli segnala che è pronta ad inviargli un dato. Il microprocessore 6510 controlla allora (mediante interrupt o per semplice sondaggio) se il bit di interrupt FLAG è posto ad 1, e allora va alla ricerca del dato che gli proviene dalla periferica. Questa sa a sua volta che può inviarne eventualmente un altro quando è presente il segnale PC visto sopra.

4.3.2 I Timer

Il 6526 comprende due timer di 16 bit ciascuno. Essi possono quindi contare sino a 65536. Ciascuno di essi può venire letto, ed il valore che

si ha in risposta corrisponde al suo stato corrente.
Inoltre i dati (ad esempio il valore iniziale) possono venire scritti entro dei buffer speciali che potranno modificare i valori dei timer.
Essi possono venire usati da soli, oppure collegati fra di loro per delle applicazioni che comportano un conteggio di maggiore durata.
Il funzionamento di questi timer ed il loro diverso modo operativo sono gestiti dai registri di controllo CRA e CRB. Il registro CRA si riferisce al timer A, e quello CRB al timer B. Tali registri sono di 8 bit ciascuno.
Esamineremo questi due registri bit per bit, descrivendo contemporaneamente le diverse caratteristiche dei due timer.

1. Registro CRA

Bit 0. Questo bit serve per far partire o per arrestare in ogni istante il Timer A. Quando vale 1 il timer funziona, mentre quando è a 0 è fermo.

Bit 1. Posizionando questo bit ad 1 si può disporre su PB6 del segnale di uscita del Timer A, anche se questo piedino è configurato in entrata dal registro DDRA. Quando invece si trova a 0 siamo in modo di funzionamento normale.

Bit 2. Questo bit permette di determinare l'andamento del segnale generato su PB6 quando è selezionato questo modo operativo (vedi Bit 1).
Quando tale bit è posto a 0 si è in modo "listabile"; ogni volta che il timer passa per lo zero l'uscita PB6 cambia di stato (da 0 ad 1 o da 1 a 0).
Il timer viene allora ricaricato con il valore contenuto nel buffer cui abbiamo accennato prima. Quando il Timer viene messo in marcia per la prima volta, l'uscita corrispondente si pone automaticamente nello stato 1. Nel caso in cui il bit 2 viene posto ad 1, si è in modo "generazione di impulsi": ogni volta che il timer passa per lo zero, sull'uscita PB6 viene applicato un impulso positivo singolo, della durata di un ciclo di temporizzazione.

Bit 3. Questo bit consente di selezionare il funzionamento "a colpo singolo" oppure "continuo". Quando il bit si trova posto ad 1, viene selezionato per il timer A il modo "ad un colpo": il timer decrementa il conteggio a partire dal valore contenuto nel buffer, sino a raggiungere il valore 0. A questo punto genera un interrupt, poi si ricarica nuovamente assumendo il valore contenuto nel buffer, e ferma il conteggio. In modo "continuo" (Bit 3 = 0), il timer ripete la procedura citata ogni volta, sino a quando non viene arrestato dal bit 3 posto a 0.

Bit 4. Questo bit serve per il caricamento del buffer del Timer A, che può avvenire in un istante qualsiasi, mentre il conteggio è in corso oppure no. Il caricamento sarà forzato quando tale bit è posto ad 1.

Bit 5. Il Timer A può contare (o meglio, decrementare di valore) in ragione di un'unità per ciclo di temporizzazione, oppure per azione di un segnale esterno applicato ad un piedino CNT (sulla porta utene sono disponibili CNT1 e CNT2). Quando tale bit vale 0, il timer conta i cicli di clock Φ_2 generati dal 6510, internamente. Quando tale bit vale 1, esso conta le transizioni negative (passaggio da 1 a 0) del segnale applicato all'ingresso CNT.

Bit 6. Questo bit si riferisce al registro e spostamento impiegato per le entrate/uscite seriali. Lo consideremo un po' più avanti in questo stesso capitolo.

Bit 7. Questo bit si riferisce all'orologio integrato del 6510, e sarà anch'esso descritto più oltre.

2. Registro CRB

Bit da 0 a 4. I bit di questo registro hanno funzioni del tutto simili a quelle già descritte per il registro CRA, con l'eccezione che l'uscita del Timer B compare sul piedino PB7 (invece del PB6 nel caso del CRA).

Bit 5 e 6. Questi bit permettono di selezionare il segnale sotto la cui azione il Timer B effettua il (de)conteggio.
La tabella che segue riassume i risultati ottenuti in funzione dei valori assunti dai bit 5 e 6:

Bit 5	Bit 6	Funzione
0	0	Il Timer B conteggia i cicli di temporizzazione (Φ_2)
0	1	Il timer conta le transizioni negative del segnale applicato sul piedino CNT
1	0	Il Timer B conta gli impulsi generati dal Timer A quando passa per lo zero
1	1	Il Timer B conta gli impulsi generati dal Timer A quando passa per lo zero, ma soltanto se il piedino CNT si trova a 0

Quando il bit 5 è posto ad 1 si possono collegare i due Timer A e B allo scopo di effettuare conteggi più lunghi.

Bit 7. Questo bit è utilizzato per l'allarme ("sveglia"), e verrà descritto più avanti.
Ed ora vediamo qualche esempio di applicazione per i due timer. Supponiamo di voler generare un tempo di ritardo di 65 millisecondi. Sarà sufficiente scrivere il programmino seguente:

```
C000>LDA # 08     ; modo "a un colpo"
C002>STA DD0E
C005>LDA # FF     ; carica il buffer del Timer A
C007>STA DD04     ; byte basso
C00A>STA DD05       byte alto
C00D>LDA # 09     ; fa partire il contatore
C00F>STA DD0E
C012>LDA DD0D     ; tempo trascorso?
C015>AND # 01
C017>BEQ F9       ; no, riprendi
C019>RTS          ; sì, terminato
```

Il programma funziona a questo modo:

— Per prima cosa, si seleziona il modo di conteggio "ad un colpo", ponendo ad 1 il bit 3 del registro CRA.
— I buffer relativi al Timer A vengono caricati con il valore corrispondente a 65 millisecondi circa (esattamente, 65536 cicli di temporizzazione).

La determinazione della fine del conteggio si effettua qui tramite il registro di controllo degli interrupt, sul quale torneremo più avanti.
Diciamo comunque che quando il tempo è trascorso (passaggio per lo 0 del timer), il bit 0 del registro ICR viene posto ad 1.
Ora supponiamo di voler generare un ritardo più lungo (17 secondi). Per fare ciò, dobbiamo collegare fra di loro i due timer del circuito CIA2.

Il programma corrispondente è il seguente:

```
C000 >LDA # 00     ; Timer A: modo "continuativo"
C002 >STA DD0E
C005 >LDA # 48     ; Timer B: modo "ad un colpo", collegato
C007 >STA DD0F       al Timer A
C00A>LDA # FF      ; carica il Timer A (65536 cicli di clock)
C00C>STA DD04      ; byte basso
C00F>STA DD05      ; byte alto
C012 >STA DD06     ; carica il Timer B (256 passaggi per lo 0 di
C015 >LDA # 00       A)
C017 >STA DD07
C01A>LDA # 01      ; fa partire il Timer A
C01C>STA DD0E
C01F>LDA # 49      fa partire il Timer B
C021 >STA DD0F
C024 >LDA DD0D     ; tempo trascorso?
C027 >AND # 02
C029 >BEQ F9       ; no, riprendi
C02B>RTS           ; sì, terminato
```

Anche il funzionamento di questo programma è assai semplice:

— Il Timer A viene posto in modalità "conteggio continuativo" (bit 3 di CRA = 0).
— Il Timer B viene invece posto in modo "ad un colpo" (bit 3 di CRB = 1), e viene programmato in modo che effettui il conteggio dei passaggi per lo zero del timer A (bit 5 = 1, bit 6 = 0 in CRB).

Il Timer A viene caricato con un valore che corrisponde ad un conteggio di 65536 cicli di clock ($FF in TAL e $FF in TAH).
Il Timer B viene invece caricato con un valore corrispondente a 256 passaggi del Timer A per lo zero ($FF in TBL); il tutto corrisponde a un ritardo totale di circa 17 secondi (16.777.216 cicli di clock). La fine del conteggio viene verificata controllando il valore del bit 2 del registro ICR, che passa ad 1 quando il Timer B passa per lo 0.
Nell'ultimo esempio che consideremo, vogliamo generare una serie ("treno") di impulsi sul piedino PB6. La frequenza relativa deve essere di 1 kHz, il che corrisponde ad una larghezza degli impulsi di 500 microsecondi. Per fare questo basterà conteggiare circa 500 cicli di clock.

Ecco il programma corrispondente:.

```
LDA>#06           ; modo "conteggio continuativo"; uscita su
                    PB6
STA >DD0E         ; Timer A
LDA > # F4        ; carica il timer: 500 cicli di clock
STA >DD04
LDA> # 01
STA>DD05
LDA># 07          ; fa partire il treno di impulsi
STA> DD0E
```

Il programma è tanto semplice da non richiedere alcun commento.

4.3.3 L'"orologio"

Il vostro Commodore 64 possiede già un orologio in tempo reale, di cui abbiamo parlato nel 1° volume e nel capitolo relativo ai sottoprogrammi presenti nella ROM Monitor del presente libro.
In effetti, la nostra macchina è dotata di tre "orologi", perché ciascun circuito tipo 6526 ne possiede uno integrato. Essi non sono utilizzati dal sistema operativo del Commodore 64, ed ecco perché potete impiegarli per i vostri scopi. Alla fine di questo paragrafo vi forniremo un programma che vi permetterà di accedere in qualsiasi momento ad un orologio in tempo reale assai preciso (non può dirsi altrettanto per quello già presente in origine sulla macchina).
Ma prima di questo, vediamo un momento come accedere a questa assai interessante caratteristica del circuito 6526.
Abbiamo visto che esistono 4 registri, chiamati TOD 10, TOD SEC, TOD MIN e TOD HRS che si riferiscono a questo orologio.
L'orologio del 6526 indica quindi le ore, i minuti, i secondi ed i decimi di secondo. Esso funziona su un ciclo di 12 ore, ed il bit superiore del registro TOD HSR indica se si tratta di "mattino" o "pomeriggio" (AM/PM in inglese).
L'ora viene fornita in formato BCD (un byte di 8 bit comprende due cifre comprese fra 0 e 9, ossia fra 0000 e 1001).
Questo orologio impiega come riferimento la frequenza di 50 Hz della rete, cosa che gli permette di avere un'ottima precisione a breve termine, ed una eccellente precisione a lungo termine.
Esso può venire programmato (ossia rimesso all'ora), oppure venire letto. Inoltre esso possiede un "allarme' incorporato capace di generare

un interrupt al momento desiderato.
Oltre ai quattro registri detti sopra, esistono due bit di controllo in CRA e CRB che permettono di selezionare le diverse funzioni seguenti:

Registro CRA: bit 7. Nel caso di funzionamento in Europa (e da noi in Italia in particolare), questo bit dovrà essere posto ad 1, perché così si seleziona il modo di funzionamento a 50 Hz. (Se tale bit si trova invece a 0, viene selezionato il modo a 60 Hz, valido per gli USA).

Registro CRB: bit 7. Questo bit permette di programmare l'ora ovvero il tempo dell'allarme.
Quando vale 0, si accede ai registri per la rimessa all'ora dell'orologio.
Quando vale 1, si accede ai registri per il posizionamento dell'ora dell'allarme.
Ed ora descriveremo la procedura da seguire per la rimessa all'ora dell'orologio, o per leggerne il tempo.

1. Lettura

Per leggere il tempo segnato dall'orologio, basterà leggere il contenuto dei diversi registri TOD. (Occorre preventivamente mettere a 0 il bit CRB7). Così si leggono di seguito le ore, i minuti, i secondi ed i decimi di secondo. Quando viene letto il contenuto di TOD HRS, il tempo dell'orologio viene memorizzato, sino a quando si leggono i decimi di secondo.
In tal modo si evitano errori di lettura.
La tabellina seguente riassume le diverse informazioni disponibili nei registri TOD:

	Bit 0-3	Bit 4-7
TOD 10	decimi di secondo	0 0 0 0
TOD SEC	secondi (unità)	secondi (decine)
TOD MIN	minuti (unità)	minuti (decine)
TOD HRS	ore (unità)	ore (decine: 0 e 1) bit 7 = 0: mattina 1: pomeriggio

Per leggere l'orologio con un programma in L.M. si scriverà:

```
C000 >LDA # 80      ; modo 50 Hz
C002 >STA DD0E
C005 >LDA # 00      ; accesso all'orologio
C007 >STA DD0F
C00A >LDA DD0B      ; salva le ore
C00D >STA C023
C010 >LDA DD0A      ; salva i minuti
C013 >STA C024
C016 >LDA DD09      ; salva i secondi
C019 >STA C025
C01C >LDA DD08      ; salva i decimi di secondo
C01F >STA C026
C022 >RST
```

2. Scrittura

Per la rimessa all'ora dell'orologio basterà analogamente scrivere entro i diversi registri TOD.
Per prima cosa si inseriranno il valore delle ore, poi dei minuti, poi dei secondi ed infine i decimi di secondo.
L'orologio incomincia a funzionare solo nell'istante in cui è stato caricato quest'ultimo parametro, il che assicura una buona precisione.
Per la messa all'ora dell'orologio sul tempo: 08h 25m 53s 9/10 si scriverà:

```
C000 >LDA # 80      ; modo 50 Hz
C002 >STA DD0E
C005 >LDA # 00      ; accesso all'orologio
C007 >STA DD0F
C00A >LDA # 08      ; messa all'ora dell'orologio: 8 ore (BCD)
C00C >STA DD0B
C00F >LDA # 25      ; 25 minuti (BCD)
C011 >STA DD0A
C014 >LDA # 53      ; 53 secondi (BCD)
C016 >STA DD09
C019 >LDA # 09      ; 9 decimi di secondo (BCD)
C01B >STA DD08
C01E >RST
```

Per accedere all'allarme, le operazioni da effettuare sono esattamente le stesse, salvo che il bit CRB7 dovrà essere stato prima posto ad 1.
Il bit 2 del registro ICR verrà dunque posto ad 1 quando l'orologio raggiungerà il valore programmato nei registri riservati al tempo dell'allarme. Notiamo che con questo allarme si può generare un interrupt (interruzione), e così avviare una sirena o un qualsiasi altro dispositivo di vostra scelta. Torneremo su questo punto nel paragrafo concernente gli interrupt. Ed ora, come accennato, vogliamo darvi un esempio di applicazione di questo tipo di orologio.
Il nostro intento è di realizzare un orologio in tempo reale che sia più semplice da usare, e più preciso, di quello già disponibile sul Commodore 64 con le istruzioni BASIC TI e TI$.
In pratica i limiti che questo presenta sono i seguenti:

— Esso arresta il conteggio quando viene caricato o salvato un file od un programma
— Se un programma è in corso di esecuzione, occorre fermarlo, domandare che venga visualizzata l'ora, e poi proseguire impostando CONT.
— Il suo errore può raggiungere qualche minuto nelle 24 ore
— Infine, per avere l'ora, occorre chiederlo al computer, mentre sarebbe senz'altro comodo vederla continuamente visualizzata sullo schermo, in alto a destra per esempio.

L'orologio che abbiamo realizzato pone rimedio a tutti questi problemi. Esso fa appello ad un programma in Assembler che occupa poco spazio, e può venire richiamato da un qualsiasi programma BASIC.
L'ora può venire visualizzata in permanenza sullo schermo, senza interferire in alcun modo con lo svolgimento di un altro programma, perché la relativa gestione è per interrupt.
Se si vuole, si può anche per comodità fare interrompere la scansione dei decimi di secondo. o anche dei secondi, per via interna.

Ecco il listato del programma in BASIC, che contiene pure il L.M.:

```
 10 REM PROGRAMMA OROLOGIO
 20 DATA 120, 173, 20, 3, 162, 89, 141, 224, 3, 142, 20, 3, 173, 21, 3
 30 DATA 162, 3, 141, 225, 3, 142, 21, 3, 88, 96, 173, 11, 220, 170, 41
 40 DATA 15, 24, 105, 48, 141, 67, 4, 138, 16, 4, 162, 16, 16, 2, 162, 1,
         142
 50 DATA 77, 4, 162, 32, 41, 16, 240, 2, 162, 49, 142, 66, 4, 173, 10,
         220
 60 DATA 170, 41, 15, 105, 48, 141, 70, 4, 138, 74, 74, 74, 74, 24, 105,
         48
 70 DATA 141, 69, 4, 173, 9, 220, 170, 41, 15, 105, 48, 141, 73, 4, 138,
         74
 80 DATA 74, 74, 74, 24, 105, 48, 141, 72, 4, 173, 8, 220, 105, 48, 141,
         75
 90 DATA 4, 169, 32, 141, 65, 4, 141, 76, 4, 141, 79, 4, 162, 15, 157, 24
100 DATA 4, 202, 208, 250, 169, 58, 141, 68, 4, 141, 71, 4, 169, 46, 141,
         74
110 DATA 4, 169, 13, 141, 78, 4, 169, 162, 13, 157, 65, 216, 202, 208,
         250, 76, 0,0
120 DATA 120, 173, 224, 3, 141, 20, 3, 173, 225, 3, 141, 21, 3, 88, 96
130 C0 = 0
140 FOR I=832 TO 1008
150 READ A: POKE I,A: C0 = C0+A : NEXT
160 IF C0 — 15605 THEN PRINT "ERRORE!":STOP
170 REM INIZIALIZZAZIONE
180 INPUT "ORE"; H
190 INPUT "MINUTI"; M
200 P=0:IF H > 12 THEN H=H—12:P=1
210 IF H > 9 THEN H=H+6
220 IF P=1 THEN H=H+128
230 POKE 56331, H: POKE 56329, 0
240 M=INT(M/10) * 6 + M : POKE 56330, M : POKE 56328, 0
250 SYS(832) : SYS(994)
260 END
```

Impostate completamente il programma, e poi battete RUN. Se avete fatto qualche errore nell'impostazione di qualche DATA, il programma si fermerà da solo ed emetterà il messaggio "ERRORE".
Poi impostate le ore ed i minuti del tempo a cui volete inizializzare l'orologio. Dopo aver battuto i minuti seguiti da RETURN, l'orologio

si mette automaticamente all'ora indicata. Per vederla in funzione, battete o chiamate da programma BASIC SYS (832).
Potete arrestare la visualizzazione eseguendo SYS (996) seguito dalla cancellazione dello schermo con SHIFT + CLEAR/HOME. L'orologio continuerà a marciare internamente. In qualsiasi momento potete riprendere la visualizzazione con SYS(832). Osserviamo che la routine in L.M. che realizza la funzione orologio è collocata fra gli indirizzi 832 e 1008, posizione del buffer impiegato dal lettore di cassette durante i salvataggi o i caricamenti. La routine verrà quindi distrutta se viene eseguita una di tali operazioni.
Se lo desiderate, potete quindi memorizzare i dati del L.M. contenuti nei DATA in un'altra collocazione di memoria, a condizione di modificare opportunamente la routine stessa del L.M..

4.3.4 Le comunicazioni seriali

Queste avvengono per il tramite dei piedini SP1 ed SP2 della porta utente, e possono avvenire sia in entrata che in uscita.
La selezione fra questi due modi avviene tramite il bit 6 del registro CRA. Se esso è posto ad 1, si è in uscita; se è posto a 0, in entrata. Esaminiamo separatamente queste due modalità.

1. Modalità entrata

In questo modo le informazioni seriali vengono applicate al piedino SP del 6526 (SP1 e SP2 sulla porta utente).
Successivamente esse vengono trasmesse e "spostate" entro il registro SDR (registro per le comunicazioni seriali), ad un ritmo stabilito dal segnale applicato al piedino CNT. Quando sono stati acquisiti (ricevuti) 8 bit (l'acquisizione di un bit si fa sui fianchi discendenti del segnale applicato su CNT), viene generato un interrupt (il bit 3 del registro ICR viene posto ad 1).

2. Modalità uscita

In questo modo le informazioni presenti nel registro SDR vengono trasmesse serialmente al piedino SP, ad una velocità fissata dal contenuto del Timer A (un passaggio allo zero ogni due). La trasmissione inizia dopo un'operazione di scrittura entro il registro SDR, ed il Timer A deve venir programmato per funzionamento "continuativo".

4.3.5 Gli Interrupt (interruzioni)

Abbiamo visto come i timer, ad es. l'orologio, sono capaci di posizionare in certi casi un dato bit del registro ICR. Simultaneamente, il 6526 è capace di generare una "interruzione" (viene comunemente usato il termine inglese "interrupt"), a condizione che questa risulti abilitata: avviene allora un salto ad un sottoprogramma in L.M. speciale.
Esistono quindi due registri, entrambi chiamati ICR ma dei quali uno può essere usato solo in lettura, e l'altro unicamente in scrittura. Quest'ultimo permette di interdire (disabilitare) o di abilitare gli interrupt provenienti dalle diverse fonti possibili (timer, orologio, ...).
La tabella che segue raggruppa le diverse funzioni realizzate dai vari bit dei registri ICR.

1. ICR (di sola lettura)

Bit n°	Funzione
0	Timer A: passaggio per lo 0 (bit 0 = 1)
1	Timer B: passaggio per lo 0 (bit 1 = 1)
2	Allarme: ora dell'allarme: tempo indicato dall'orologio (bit 2 = 1)
3	Porta seriale: registro SDR pieno (bit 3 = 1)
4	Piedino FLAG: transizione negativa (bit 4 = 1)
5	non utilizzato
6	non utilizzato
7	BIT IR: posto ad 1 quando interviene un interrupt abilitato

2. ICR (di sola scrittura)

Bit n°	Funzione
0	Bit 0 = 1: abilita un interrupt da parte del Timer A
1	Bit 1 = 1: abilita un interrupt da parte del Timer B
2	Bit 2 = 1: abilita un interrupt all'ora di allarme programmata
3	Bit 3 = 1: abilita un interrupt quando il registro SDR è pieno
4	Bit 4 = 1: abilita un interrupt quando si ha una transizione negativa sul piedino FLAG
5	non utilizzato
6	non utilizzato
7	Bit 7 = 0: rimette a zero i bit da 0 a 4 Bit 7 = 1: pone ad 1 i bit desiderati

Osserviamo che quando interviene un'interruzione autorizzata (abilitata), il bit IR del registro ICR viene "settato" (posto ad 1), mentre il piedino IRQ del 6526 passa allo stato basso, il che ha per effetto di far saltare il 6510 al sottoprogramma speciale per la gestione degli interrupt. Non diremo altro a proposito di tale circuito che, nonostante la sua complessità, risulta tuttavia relativamente facile da programmare (a paragone per esempio con il 6522 che è utilizzato sul VIC 20).
Ed ora passeremo ad affrontare l'ultimo dei capitoli di questo libro, che passa in rassegna le diverse unità periferiche ed i programmi pronti disponibili sul mercato per il Commodore 64, allo scopo di esservi di aiuto per un'eventuale scelta.

5
Le unità periferiche ed i programmi pronti del Commodore 64

Il Commodore 64 è stato concepito per poter venire usato tal quale (salvo l'ovvio collegamento ad un monitor o ad un apparecchio TV), ma mantenendo le più ampie possibilità di estensione, sia per il tramite del connettore per le espansioni, che tramite la porta seriale o la porta utente (senza parlare delle porte di controllo, destinate al collegamento delle paddle o dei joystick per i giochi).
Ecco perché la Commodore (così come vari altri costruttori) propone una gamma assai vasta in unità periferiche e di programmi pronti studiati particolarmente per l'impiego con il Commodore 64.
Va osservato che la maggior parte delle periferiche utilizzabili con il VIC 20 sono compatibili con il Commodore 64, il che rende relativamente semplice ed economico il passaggio dall'una macchina all'altra.

5.1 Le unità periferiche

5.1.1 Il registratore a cassette

Questo è il modello COMMODORE C2N, e si tratta probabilmente della prima unità periferica esterna che avete acquistato assieme alla vostra macchina (a meno che non abbiate optato direttamente per l'acquisto di un'unità disco).
Esso viene alimentato direttamente dal Commodore 64, e si collega sul retro della vostra macchina tramite un connettore speciale.
Il funzionamento del Gruppo Commodore 64 + registratore a cassette è relativamente interattivo. In pratica, è il computer a comandare il mo-

vimento del nastro, mentre provvede pure ad emettere messaggi tipo

PRESS PLAY ON TAPE quando legge un nastro

ovvero

PRESS RECORD AND PLAY ON TAPE quando deve registrare

La registrazione sulla cassetta può riguardare programmi in BASIC o anche programmi in linguaggio macchina, nonché semplici dati (byte). Per i programmi BASIC si impiegano le istruzioni SAVE e LOAD, secondo le procedure già descritte nel 1° volume.
Per i programmi in linguaggio macchina, potete fare riferimento al capitolo a questi dedicato in questo 2° volume.
La gestione dei file di dati si fa tramite le istruzioni OPEN, CLOSE, PRINT#, GET#, INPUT#, anche questi già visti e descritti nel 1° libro.

5.1.2 L'unità disco

Acquistando un'unità disco voi avete accresciuto considerevolmente la potenza del vostro sistema, perché in tal modo potete disporre di una capacità di memoria di 174 K byte per disco. L'unità disco è prevista per il collegamento sulla porta seriale del Commodore 64, e reca il numero di riferimento VIC 1541.
Il suo principale vantaggio è di essere una periferica, come si suol dire, "intelligente", perché dotata di un proprio microprocessore che si incarica di svolgere operazioni quali

— la lettura di un disco
— la scrittura su di un disco
— la gestione del "directory" (catalogo dei file)
— la formattazione od inizializzazione del disco
— la gestione dei file di tipo sequenziale o ad accesso diretto
— la trasmissione di dati tramite il canale seriale, ecc.

Il programma che permette la gestione completa di queste operazioni si chiama DOS (Disk Operating Sytstem), ed è contenuto nella ROM.
Il maggiore inconveniente legato a questo tipo di periferica è che il collegamento fra di essa ed il computer è relativamente poco veloce (per-

ché è del tipo seriale), il che può diventare veramente fastidioso nel caso di lunghi programmi.
I trasferimenti basilari fra il Commodore 64 e l'unità disco avvengono sotto la regia dei comandi LOAD, SAVE, OPEN, CLOSE, INPUT#, PRINT#, GET#.
Le loro funzioni generali sono state descritte nel 1° volume.
Per maggiori particolari, ad esempio riguardo la gestione dei file, potete consultare il Manuale d'uso che vi è stato fornito assieme all'unità disco, che contiene tutte le informazioni necessarie.

5.1.3 La stampante

La stampante, del modello VIC 1525, è prevista per venire impiegata con il Commodore 64 tramite l'interfaccia seriale. Se diverse periferiche utilizzano contemporaneamente questo bus (ad esempio due stampanti, più un disco, ecc...), esse devono essere collegate in cascata.
Nel caso della trasmissione di dati, ad esempio, la periferica interessata riconoscerà da sola se questi sono destinati a se stessa, grazie all'associazione file binario/numero di periferica, visto nel 1° volume.
Questa stampante è del tipo "ad aghi". Essa può venire impiegata in vari modi:

— *Modo normale o standard*: la stampante imprime del testo, di cui ciascun carattere è definito da una matrice di 5×7 punti (80 caratteri per riga). Possono essere stampate sia lettere minuscole che maiuscole.
— *Modo grafico*: la stampante imprime allora dei grafici, punto per punto. La stampa comprende allora un massimo di 480 colonne, ciascuna colonna essendo costituita da 7 punti.

Esempio

```
OPEN 1,4
```

permette di selezionare la stampa di caratteri maiuscoli o grafici

```
OPEN 1,4,7
```

permette di stampare minuscole o maiuscole.
Si possono ottenere funzioni speciali inviando dei codici tipo CHR$.

Ad esempio:

CHR$(8)	passaggio in modo grafico
CHR$(10)	Line Feed = interlinea
CHR$(13)	Carriage Return = ritorno a capo
CHR$(14)	caratteri di doppia larghezza
CHR$(15)	ritorno ai caratteri normali
CHR$(16)	tabulazione
CHR$(17)	passaggio al set di caratteri maiuscole/minuscole
CHR$(18)	caratteri inversi
CHR$(26)	ripetizione dell'ultimo dato grafico
CHR$(27)	spostamento nel punto considerato
CHR$(145)	passaggio al set di caratteri maiuscoli/semigrafici
CHR$(146)	fine del modo caratteri inversi

Per maggiori particolari riguardo l'impiego di questi comandi, potete fare riferimento al manuale fornito assieme alla vostra stampante.

5.1.4 Gli accessori per i giochi

Il Commodore 64 dispone di due porte di controllo, situate sul lato destro della macchina. Esse sono destinate al collegamento dei paddle, dei joystick o di una penna ottica (Light Pen). Quest'ultima può essere collegata solo alla porta A.
I joystick si collegano sul circuito di controllo degli I/O CIA1 tipo 6526. Come abbiamo visto nel capitolo precedente, la porta A si trova all'indirizzo 56320 (=$DC00), e la porta B all'indirizzo 56321 (=$DC01).
Un joystick si compone di 5 interruttori (4 per le diverse direzioni ed 1 per il "fuoco").
Gli interruttori corrispondono ai 5 bit inferiori delle porte A e B citate. Normalmente, se il pulsante di "fuoco" non è premuto, o se non è stata ancora selezionata alcuna direzione col movimento della levetta, il bit corrispondente sarà ad 1. Altrimenti esso passa a 0.
Il ruolo dei bit citati è il seguente:

Bit 0 = interruttore 0 = direzione verso l'alto
Bit 1 = interruttore 1 = direzione verso il basso

Bit 2 = interruttore 2 = direzione verso sinistra
Bit 3 = interruttore 4 = direzione verso destra
Bit 4 = interruttore 4 = pulsante di "sparo"

Queste indicazioni dovrebbero essere sufficienti per poter programmare i vostri giochi od altri programmi con l'uso dei joystick. Maggiori dettagli per la programmazione del circuito CIA1 li potete trovare nel capitolo precedente.
I paddle, invece, sono collegati contemporaneamente al circuito CIA1 ed al circuito di controllo del suono.
I valori digitalizzati dei potenziometri che costituiscono i paddle possono leggersi nei registri di indirizzo 54297 (=$D419) e 54298 (=$D41A).
La posizione sullo schermo determinata dalla penna ottica è memorizzata agli indirizzi 53267 (=$D013) e 53268 (=$D014). Quest'ultimo registro corrisponde all'ordinata Y, mentre l'ascissa X è data in 53267.
Va notato che l'impiego dei joystick, paddle e Light Pen passa quasi obbligatoriamente attraverso la scrittura di routine in linguaggio macchina. Infatti l'impiego del BASIC, che risulta troppo lento, non permetterebbe di ottenere risultati abbastanza accurati, soprattutto per quanto riguarda i joystick o i paddle, ad esempio nel caso dei videogiochi.

5.1.5 La cartuccia Z/80

È possibile collegare al connettore per le espansioni del Commodore 64 una speciale cartuccia munita di un microprocessore Z/80. Quando viene selezionato questo microprocessore, la ROM BASIC non è più accessibile. Si dispone allora di 64 K byte di memoria "attiva" (RAM), che contengono il DOS (od il CP/M all'occorrenza) ed i programmi utente (ossia quelli scritti da voi stessi o quelli acquistabili in commercio: altri linguaggi, trattamento di testi (Word Processing, ecc.).
Per poter lavorare sotto CP/M con la cartuccia Z/80 si richiede almeno un'unità dischi. Inoltre, per poter impiegare i programmi che operano con questo sistema operativo, occorre che il fabbricante li abbia espressamente scritti nel formato per il Commodore 64, che non è obbligatoriamente quello adottato dai vari fabbricanti.
La disponibilità di questi programmi nel nostro Paese non è certa, per cui dovrete prestare attenzione prima di acquistare programmi che operano in regime CP/M.
La lista delle periferiche e delle espansioni riportate sopra non ha ovvia-

mente alcuna pretesa di risultare completa. Esistono sul mercato numerose schede (ad es. scheda 80 colonne), fabbricate negli Stati Uniti ed anche altrove. Probabilmente queste espansioni sono già o saranno presto disponibili anche da noi.

5.2 I programmi pronti

I programmi pronti disponibili per il Commodore 64 sono di tre tipi:

— cassette impiegabili con il registratore tipo C2N
— cartucce collegabili sulla porta di espansione
— dischetti, che richiedono ovviamente un'unità di lettura dischi, tipo VIC 1541.

Le cassette di nastro hanno il vantaggio di costare poco, ma hanno il difetto di essere lunghe da caricare.
Le cartucce di espansione ("cartridge") mettono a disposizione dell'utilizzatore una biblioteca assai vasta di programmi pronti per l'impiego (che non richiedono alcun caricamento), e che partono automaticamente.
I dischetti necessitano come detto di una unità disco (VIC 1541), periferica abbastanza costosa; inoltre non ve ne sono ancora molti sul mercato.
I programmi attualmente disponibili, od annunciati, dalla Commodore per il C 64 si possono raggruppare in tre categorie:

1. Giochi

In genere sotto forma di cartucce di espansione, richiedendo spesso l'uso dei joystick o dei paddle.

2. Programmi di utilità

Esistono al momento, sotto forma di cassette o di cartucce di espansione, vari programmi che consentono in uso più comodo del Commodore 64. Possiamo citare fra gli altri:

* Monitor in linguaggio macchina

Esiste una cartuccia fabbricata dalla Commodore che permette di programmare il Commodore 64 in L.M. Data questa forma di commercializzazione, non richiede alcun tempo di caricamento: in cambio risulta di costo elevato. Per conto nostro, abbiamo messo a punto una cassetta assai più economica che possiede le stesse caratteristiche (ed altre ancora in più) della cartuccia 64MON. Fra i comandi disponibili citiamo:

— *Assemble*: permette di convertire delle linee scritte in Assembler 6510 (mnemonici) in codice macchina eseguibile. Questo comando accetta anche i "label", il che permette un assemblaggio quasi automatico dei programmi comprendenti varie istruzioni di salto.
— *Disassemble*: che consente di "disassemblare" il contenuto di una data parte della memoria (RAM o ROM) del Commodore 64.
— *Irace*: molto utile per la messa a punto dei programmi; permette di visualizzare il contenuto dei registri nel corso dell'esecuzione passo-passo d'un programma in L.M.
— *Punto di arresto* ("Breakpoint"): permette di arrestare l'esecuzione di un programma ad un punto predeterminato, e di controllare ed eventualmente modificare il contenuto dei vari registri.
— *Load e Save*: che permettono di caricare o salvare un programma in L.M. collocato in un punto qualsiasi della memoria.
— ecc.

(N.d.T.: Non è certo che questa cassetta sia disponibile anche in Italia)

* Programmazione di note e melodie musicali

Se avete avuto l'occasione di programmare dei suoni o brevi melodie sul vostro Commodore 64, vi sarete resi conto che la programmazione del circuito di controllo del suono non è cosa facile. Esistono perciò in commercio dei programmi che permettono di realizzare un'"interfaccia", liberandovi da questo compito alquanto barboso. In questo modo potete anche trasformarvi in vero e proprio "compositore" di musica elettronica.

* Programmazione degli SPRITE

Quel che abbiamo or ora detto per il circuito di controllo del suono è valido egualmente anche per il circuito di controllo dello schermo, in particolare per quanto concerne la creazione degli SPRITE.
La definizione d'uno SPRITE è cosa relativamente facile: richiede solo del tempo. Per contro, definire 100 SPRITE è quasi impossibile senza un ausilio idoneo. Ed è quel che vi può capitare se volete creare dei giochi che comportano un gran numero di "mostri" o di "navi spaziali", la cui forma varia in funzione del grado di avanzamento del gioco.
È per questo che abbiamo creato egualmente un programma chiamato "Editor di SPRITE" (disponibile, come il precedente Monitor in L.M., presso le Edizioni Eyrolles). Esso permette di generare automaticamente gli SPRITE a partire dal loro disegno sullo schermo, poi di salvarli su cassetta per l'impiego successivo. Inoltre permette di modificarli facilmente (ruotarli, spostarli, ...).

(N.d.T.: vale quanto detto più sopra per l'Italia)

3. Programmi semi-professionali

In questa categoria si possono ricordare i programmi per il trattamento di testi (Word Processing), programmi tipo VISICALC che permettono di lavorare su grandi tabelloni di dati, programmi grafici che permettono di visualizzare degli istogrammi, delle curve di andamento economico, ecc. Negli Stati Uniti esiste un gran numero di programmi di questo tipo: probabilmente alcuni sono stati importati anche da noi. Occorre prestare attenzione che la maggior parte sono in inglese, per cui possono non prestarsi all'impiego da noi, a meno che non siano stati preventivamente tradotti (caso del Word Processing, ad esempio, se vogliamo che possa accettare anche le lettere accentate, ecc.).

4. Altri linguaggi

Il linguaggio di programmazione universale per il Commodore 64 è il BASIC, che è già contenuto nella sua ROM. Esistono altri tipi di linguaggi "superiori", e sono già apparsi o stanno per apparire programmi che implementano questi linguaggi sul Commodore.

Si possono citare in particolare:

— il FORTH (sotto forma di cartucce)
— il LOGO (sotto forma di cartucce)

Ed eccoci così giunti al termine anche di questo secondo volume de "L'IMPIEGO DEL COMMODORE 64".
Speriamo che grazie ad esso vi sia diventato quasi altrettanto facile programmare in linguaggio macchina come in BASIC, e che siate così in grado di trovare altre e nuove applicazioni per il vostro microcomputer.

Appendice

Tabella riepilogativa delle istruzioni del 6502

Modo d'indirizzamento Mnemonico	IMP	IMM	ACC	ET	PZ	EIX	EIY	PZX	PZY	REL	IND	PRE	POST
ADC		69		6D	65	7D	79	75				61	71
AND		29		2D	25	3D	39	35				21	31
ASL			ØA	ØE	Ø6	1E		16					
BCC										9Ø			
BCS										BØ			
BEQ										FØ			
BIT				2C	24								
BMI										3Ø			
BNE										DØ			
BPL										1Ø			
BRK	ØØ												
BVC										5Ø			
BVS										7Ø			
CLC	18												
CLD	D8												
CLI	58												
CLV	B8												
CMP		C9		CD	C5	DD	D9	D5				C1	D1
CPX		EØ		EC	E4								
CPY		CØ		CC	C4								
DEC				CE	C6	DE		D6					
DEX	CA												
DEY	88												
EOR		49		4D	45	5D	59	55				41	51
INC				EE	E6	FE		F6					
INX	E8												
INY	C8												

Modo d'indirizzamento Mnemonico	IMP	IMM	ACC	ET	PZ	EIX	EIY	PZX	PZY	REL	IND	PRE	POST
JMP				4C							6C		
JSR				2Ø									
LDA		A9		AD	A5	BD	B9	B5				A1	B1
LDX		A2		AE	A6		BE		B6				
LDY		AØ		AC	A4	BC		B4					
LSR			4A	4E	46	5E		56					
NOP	EA												
ORA		Ø9		ØD	Ø5	1D	19	15				Ø1	11
PHA	48												
PHP	Ø8												
PLA	68												
PLP	28												
ROL			2A	2E	26	3E		36					
ROR			6A	6E	66	7E		76					
RTI	4Ø												
RTS	6Ø												
SBC		E9		ED	E5	FD	F9	F5				E1	F1
SEC	38												
SED	F8												
SEI	78												
STA				8D	85	9D	99	95				81	91
STX				8E	86				96				
STY				8C	84			94					
TAX	AA												
TAY	A8												
TYA	98												
TSX	BA												
TXA	8A												
TXS	9A												

Abbreviazioni usate nella tabella

IMP = indirizzamento implicito
IMM = indirizzamento immediato
ACC = indirizzamento via accumulatore
ET = indirizzamento esteso
PZ = indirizzamento in pagina zero
EIX = indirizzamento esteso indicizzato tramite registro X
EIY = indirizzamento esteso indicizzato tramite registro Y
PZX = indirizzamento in pagina zero indicizzato tramite X
PZY = indirizzamento in pagina zero indicizzato tramite Y
REL = indirizzamento relativo
IND = indirizzamento indiretto
PRE = indirizzamento preindicizzato indiretto (tramite X)
POST = indirizzamento postindicizzato indiretto (tramite Y)

Come usare il tuo Commodore 64

Il secondo volume dell'opera "Come usare il tuo Commodore 64" vi permetterà di scoprire i piaceri della programmazione in Assembler su questo tipo di personal computer.
Il libro si rivolge a tutti coloro che hanno potuto intravvedere le grandi possibilità offerte da questo microcomputer, ma non hanno potuto sfruttarle compiutamente per le limitazioni del linguaggio BASIC.
Questo libro illustra tutto quanto si deve sapere per poter programmare il Commodore 64 in linguaggio macchina.
Con un taglio eminentemente pratico, vi vengono descritte numerose applicazioni: animazione, programmazione e movimento degli SPRITE, i segreti dell'INPUT/OUTPUT. Il tutto è accompagnato da numerosi esempi.